

MacTech®

The Journal of Macintosh Technology and Development

Animal Crackers

Enhancing Cocoa
QuickTime Applications

by Tim Monroe

\$8.95 US
\$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.



We got
top marks
for our
software.



You can too.

Revolution™
The Solution for Software Development
In enterprise, business and education

REVOLUTION™

Limitless possibilities

With Revolution, the solutions you can create are endless.

True cross-platform development

With the click of a button, you can build applications for Macintosh (Classic and OS X), Windows, Linux, and Unix.

Increased productivity

The powerful interface builder and easy-to-use programming language speed up all the essentials of software development, leaving you with more time to focus on your project.

Databases, multimedia, Internet applications, external libraries and more

Revolution supports everything you need: SQL databases, streaming media, control of QuickTime, QTVR and graphics, CGI processing, Internet protocols, and more.

FREE Trial Version

www.runrev.com



MacUser UK - 5 mice

©2003 Runtime Revolution Limited. All rights reserved. Runtime Revolution, the Runtime Revolution logo and Revolution are trademarks of Runtime Revolution Limited, registered in the United Kingdom. All other trademarks are the property of their respective owners.

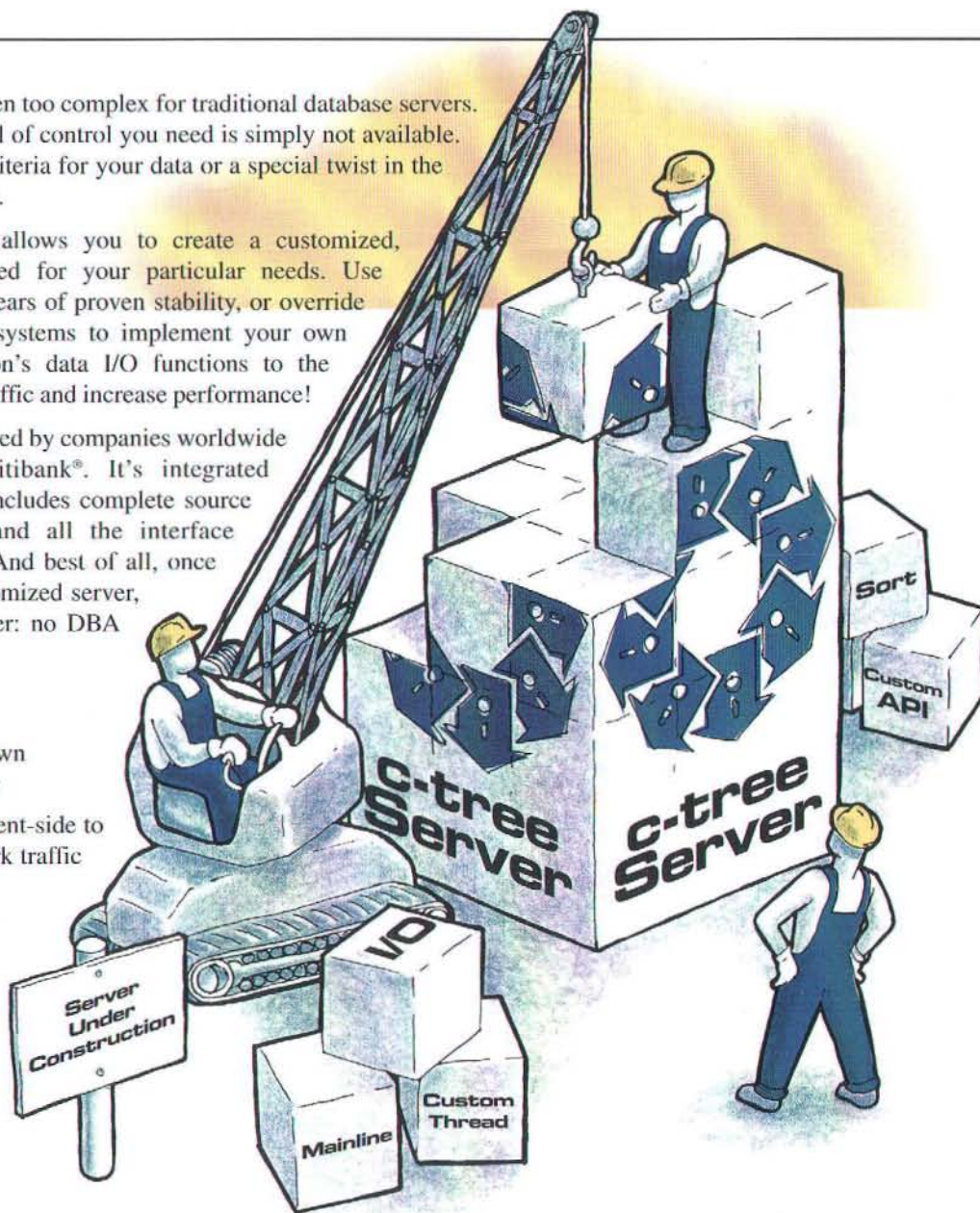
CUSTOMIZE YOUR DATABASE SERVER WITH THE C-TREE® SERVER SDK

Today's database demands are often too complex for traditional database servers. The functionality and precise level of control you need is simply not available. Perhaps you need alternate sort criteria for your data or a special twist in the threading or communication logic.

FairCom's c-tree® Server SDK allows you to create a customized, industrial-strength server designed for your particular needs. Use FairCom's kernel, with over 20 years of proven stability, or override functionality within specific subsystems to implement your own subtleties. Move your application's data I/O functions to the server-side to decrease network traffic and increase performance!

FairCom's c-tree Server SDK is used by companies worldwide such as Software AG and Citibank®. It's integrated seamlessly into c-tree Plus and includes complete source code to the server mainline and all the interface subsystems to the c-tree Server. And best of all, once you've created your unique, customized server, it is easy to install and administer: no DBA required!

- Enhance our server with your own custom server-side functionality
- Move functionality from the client-side to the server-side to reduce network traffic and increase performance
- Modify or replace entire server subsystems
- Complete source for the server mainline, key server subsystems, and client-side
- Flexible OEM licensing



Visit www.faircom.com/ep/mt/sdk today to take control of your server!



FairCom®
www.faircom.com

USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

DBMS Since 1979 • 800.234.8180 • info@faircom.com

Other company and product names are registered trademarks or trademarks of their respective owners.

© 2002 FairCom Corporation

XSERVE SCREAMS.

(FOR AN ENTERPRISE CLASS DATABASE.)

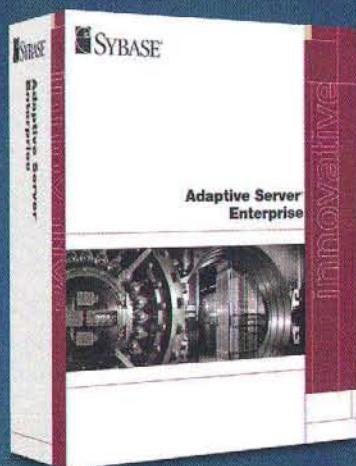


An insanely great enterprise server deserves an insanely great enterprise database. Sybase Adaptive Server™ Enterprise 12.5. Wall Street's preferred platform for mission-critical, transaction-intensive

SYBASE e-BUSINESS SOFTWARE. EVERYTHIN

THE STRAIGHT GOODS ON DATABASES.

WE ANSWER THE CALL.



enterprise applications. Now available to you on Apple Xserve™ with Mac OS X Server software. Ready to scream for joy? Check it out at www.sybase.com/mac



WORKS BETTER WHEN EVERYTHING WORKS TOGETHER.™

©2002 Sybase, Inc. All rights reserved. All trademarks are the property of their respective owners.

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail**?

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

DEPARTMENTS

Orders, Circulation, & Customer Service

Press Releases

Ad Sales

Editorial

Programmer's Challenge

Online Support

Accounting

Marketing

General

Web Site (articles, info, URLs and more...)

E-Mail/URL

cust_service@mactech.com

press_releases@mactech.com

ad_sales@mactech.com

editorial@mactech.com

prog_challenge@mactech.com

online@mactech.com

accounting@mactech.com

marketing@mactech.com

info@mactech.com

http://www.mactech.com

The MacTech Editorial Staff

Publisher • Neil Ticktin

Editor-in-Chief • Dave Mark

Managing Editor • Jessica Stubblefield

Regular Columnists

Getting Started

by Dave Mark

QuickTime ToolKit

by Tim Monroe

Reviews/KoolTools

by Michael R. Harvey

Networking

by John C. Welch

Section 7

by Rich Morin

Regular Contributors

Vicki Brown, Andrew Stone,
Erick Tejkowski, Paul E. Seving

MacTech's Board of Advisors

Jordan Dea-Mattson, Jim Straus
and Jon Wiederspan

MacTech's Contributing Editors

- Michael Brian Bentley
- Marshall Clow
- John C. Daub
- Tom Djajadiningrat
- Bill Doerrfeld, Blueworld
- Andrew S. Downs
- Richard V. Ford, Packeteer
- Gordon Garb, Sun
- Ilene Hoffman
- Chris Kilbourn, Digital Forest
- Rich Morin
- John O'Fallon, Maxum Development
- Will Porter
- Avi Rappoport, Search Tools Consulting
- Ty Shipman, Kagi
- Chuck Shotton, BIAP Systems
- Cal Simone, Main Event Software
- Steve Sisak, Codewell Corporation
- Andrew C. Stone, www.stone.com
- Chuck Von Rospach, Plaidworks
- John C. Welch

Xplain Corporation Staff

Chief Executive Officer • Neil Ticktin

President • Andrea J. Sniderman

Controller • Michael Friedman

Production Manager • Jessica Stubblefield

Production/Layout • Darryl Smart

Marketing Manager • Nick DeMello

Events Manager • Susan M. Worley

International • Rose Kemps

Network Administrator • David Breffitt

Accounting • Jan Webber, Marcie Moriarty

Customer Relations • Laura Lane
Susan Pomrantz

Shipping/Receiving • Joel Licardie

Board of Advisors • Steven Geller, Blake Park,
and Alan Carsrud

All contents are Copyright 1984-2002 by Xplain Corporation. All rights reserved. MacTech and Developer Depot are registered trademarks of Xplain Corporation. RadGad, Useful Gifts and Gadgets, Xplain, DevDepot, Depot, The Depot, Depot Store, Video Depot, Movie Depot, Palm Depot, Game Depot, Flashlight Depot, Explain It, MacDev-1, THINK Reference, NetProfessional, NetProLive, JavaTech, WebTech, BeTech, LinuxTech, MacTech Central and the MacTutorMan are trademarks or service marks of Xplain Corporation. Sprocket is a registered trademark of eSprocket Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders.

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are \$47.00 per year. Canadian subscriptions are \$59.00 per year. All other international subscriptions are \$97.00 per year. Domestic source code disk subscriptions are \$77 per year. All international disk subscriptions are \$97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

C o n t e n t s

January 2003 • Volume 19, Issue 1

MAC OS X

CamelBones

42 *Creating GUI-based apps with Perl*

By Rich Morin

GETTING STARTED

6 *A Taste of Project Builder*

by Dave Mark, Editor In Chief

MAC OS X

60 *Writing Contextual Menu Plugins for OS X, part 2*

Running Unix commands and handling text selections

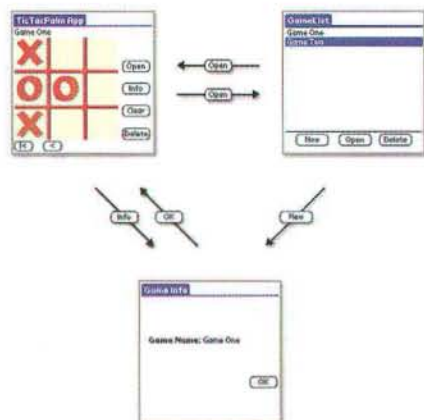
By Brent Simmons

HANDHELD TECHNOLOGIES

TicTacPalm 2

46 *Saving Data in a Palm OS Application*

by Danny Swarzman



Links Among Forms page 48

QUICKTIME TOOLKIT

24

Animal Crackers

Enhancing Cocoa QuickTime Applications

by Tim Monroe

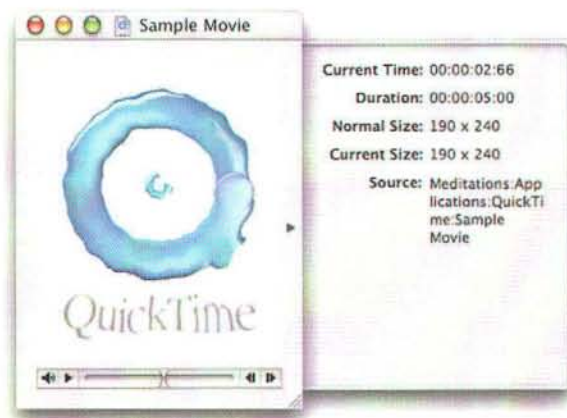
SECTION 7

12

Still More Perl

Munging Mail and Media...

by Rich Morin



A movie information drawer page 24

JAVA PROGRAMMING

Dock Tile Imaging using JDirect

6 *A pure Java approach to changing a Java application's dock tile at runtime*

By Andrew S. Downs

Copyright 2002 by Dave Mark

A Taste of Project Builder

As promised in last month's column, this month we're going to take a walk through the Project Builder debugger. Before we do, I want to touch on an issue that has been raised by a number of readers, especially folks who work in both CodeWarrior and Project Builder.

WHERE THE HECK IS THE STANDARD LIBRARY?

CodeWarrior and Project Builder each have their own distinctive look and feel, especially when it comes to the Project Window. One point of confusion concerns the location of the Standard Library. In CodeWarrior, the Standard Library is (typically) explicitly included in the project. Sometimes this is done by creating a project from a Metrowerks Standard Library (MSL) template. Other times, you add the MSL library to your project yourself, choosing from a selection of precompiled versions of the MSL, or perhaps custom compiling your own version.

Bottom line, when you look at your Project Window, you know you've got MSL in your project because it is listed in the window along with all your other source code, libraries, etc. Want to get rid of the MSL? Select it and hit the delete key.

Project Builder follows a different tack. In the Project Builder Project Window, there is no explicit reference to the Standard Library. The question people are asking is, "Where the heck is it?"

A member of Apple's Developer Tools team kindly cleared away the mist:

With gcc on Mac OS X, the "standard C library" (sometimes known as libc on Unix systems) is part of System.framework, which is implicitly brought in by gcc when linking. System.framework is a dynamic shared library, shared by all apps on the system, which reduces overall system memory use.

If the application uses C++, gcc also automatically brings in /usr/lib/libstdc++.a, which is a static library. We currently recommend that developers avoid building frameworks (shared libraries) with C++ APIs, to avoid binary compatibility problems, because gcc's

C++ ABI has been in some flux.

One other thing to note: for C & Objective-C APIs, we encourage the creation and use of frameworks. If someone does want to build a static library and link it into their app, they currently need to follow the standard Unix naming convention of lib<foo>.a. See this Q&A for more info: <http://developer.apple.com/qa/qa2001/qa1101.html>

Some of our conventions derive from wanting to help enable easy porting of Unix programs to Mac OS X; we need to balance the use of Unix conventions and Mac conventions, and try to make things seamless for everyone.

Interesting stuff. If you launch Project Builder, then click on the *Targets* tab, you'll see a list of various settings, as well as a sequence of *Build Phases* steps. **Figure 1** shows the *Frameworks & Libraries* Build Phase. The point above was that the Standard Library wasn't listed in this pane because it is built into System.framework and implicitly brought into the link process, not as an additional library added to the project.

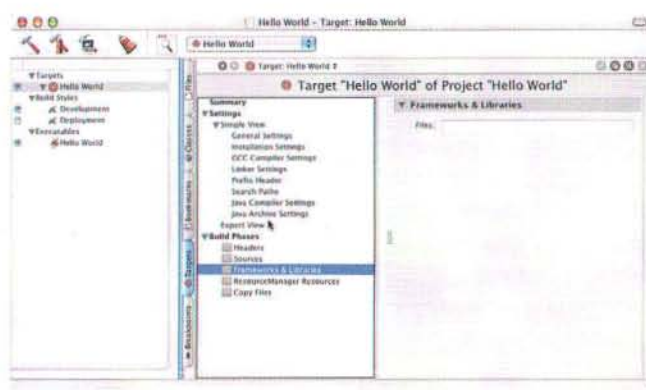


Figure 1. Project Builder's Targets pane, showing no additional Frameworks & Libraries.

Does Project Builder make use of makefiles? For you non-Unix folks, a makefile is a text file containing a script for

Dave Mark is very old. He's been hanging around with Apple since before there was electricity and has written a number of books on Macintosh development, including Learn C on the Macintosh, Learn C++ on the Macintosh, and The Macintosh Programming Primer series. Check out Dave's web site at <http://www.spiderworks.com>

Asanté Technologies – 15 years of making Mac networking ...

Faster.

Asanté built the first Mac Ethernet adapter in 1987. Today we continue to speed up networks with the new FS4100R series Fast Ethernet and GX5 series Gigabit Ethernet workgroup switches. Transfer digital content at blazing speeds over existing copper cables.



NEW!

Available for
\$299
or less!



Dedicated 100-200 Mbps data transfer pipeline for each connected client.

NEW!

Easier.

Plug-and-play solutions like our new FriendlyNET Gini memory reader/writer make moving your data easier than ever. Set up the FR1004 series cable/DSL router in minutes and begin sharing the Internet immediately – with or without wires!



NEW!

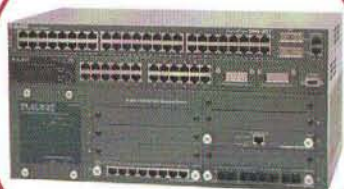
Memory cards appear as removable drives!



"The reader to buy!"
FriendlyNET Gini
Memory Reader/Writer



FriendlyNET FR1004 series cable/DSL routers – with or without 802.11b wireless



IntraCore™ 3500 series Gigabit Ethernet and FX4008 modular switches with fiber – designed for backbones, data centers and campus connectivity

Safer.

Asanté continues to develop solutions that protect your data and your investment in networking equipment. Our new IntraCore 3500 series and FX4008 modular switches combine industrial strength with extreme flexibility. FriendlyNET VR2004 series routers provide secure, redundant communications links between corporate networks and remote offices – no special drivers or software required for Macs.



NEW!

FriendlyNET VR2004 series routers – securely connect remote locations



Full support for Mac OS 9,
OS X 10.2, Apple Airport™,
WiFi, and other 802.11b-
compatible networks.



Faster. Easier. Safer.

© 2003 Asanté Technologies, Inc. Asanté and FriendlyNET are registered trademarks and AeroLAN and Gini are trademarks of Asanté Technologies. All other trademarks are property of their respective owners. All rights reserved.

Have an older Mac?
See asantestore.com
for all of your
networking needs.

building an application. Typically, a makefile will contain a series of build instructions, depending on the target being built. Project Builder's *Target* pane is, in effect, a graphical makefile. More from the Developer Tools team:

Any other frameworks and libraries besides System.framework and libstdc++ will appear in the area you show in the screen shot. The "Sources" Build Phase shows the compilation and link order of project source files.

No, there is no makefile under the interface that folks can look at. The internals of the build system are an implementation detail which we may change in the future. A developer can look at the detailed build log if they want to see the actual commands that get run during a build. To see the detailed build log, drag up the split bar at the bottom of the build pane so that the summary is at the top and the detailed log is visible below.

To get a sense of this, check out **Figure 2**. It shows the Build window from the Hello, World project with the split bars dragged wide open to reveal the Build specifics. Personally, I'd like to see this process opened up a bit more so I might tweak my compile/link instructions directly. Perhaps in a future version of Project Builder.



Figure 2. The Build window from our Hello, World project.

PLAY WITH YOUR DEBUGGER

As promised, I'd like to spend a bit of time going through Project Builder's debugger interface. Launch Project Builder, then select New Project... from the File menu. Our last Project Builder effort was a project of type "Standard Tool". This was equivalent to a Standard Library based C console app. This month, we'll create a project of type "Foundation Tool". Name the project Hellobjc and

store it in the same directory as your other projects.

The Foundation Tool project is based on the Foundation object framework and links in the Objective-C library. Over time, we'll tackle the syntax of the Objective-C language and become familiar with the classes that make up the Foundation framework. For now, let's play with this project and see if we can't learn a bit about how the debugger works.

In the project window, click on the Targets tab (the tabs are arranged vertically – sideways) and the Targets tab is fourth from the top. In the list that appears, click on the Hellobjc target (you may need to click on the Targets disclosure triangle to reveal the Hellobjc target). In the new pane that is revealed, under Settings, then under Simple View, click on the item GCC Compiler Settings (Remember last month's Terminal project? You compiled your source code with the command "gcc." GCC is the GNU C Compiler).

Make sure that the "Generate debugging symbols" check box is checked. This tells project builder to include debugging information when it compiles your code, allowing you to use the debugger to debug your program. Very important! Take a look at **Figure 3** to get a sense of what this looked like on my machine.

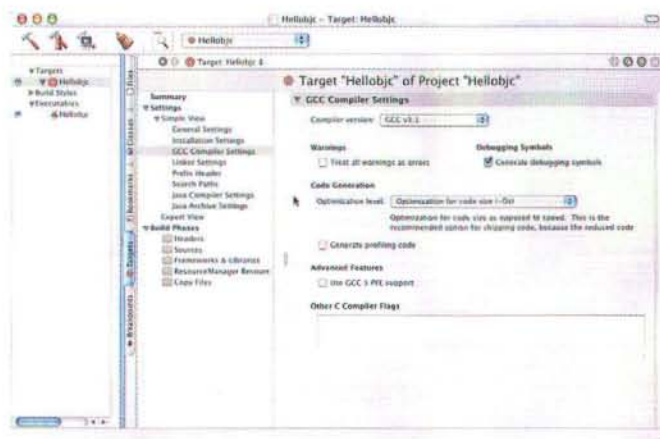


Figure 3. Make sure the "Generate debugging symbols" checkbox is checked.

Before we try out the debugger, take the program for a spin. Click on the icon with the tool-tip "Build and run active executable" (check out **Figure 4**). The Build window will appear and the program should compile and, finally, the Run window will appear showing our classic "Hello, World!" output, much as it did in the C-based Hello, World project.



Figure 4. Click on the third icon to build and run your project.

Now close the Run and Build windows, leaving the Project window open. Click on the Files tab, then select the file `main.m` (you'll find it under Source). As a reminder, Objective-C source files end with the extension `.m`. Here's the default source code from `main.m`:

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

    // insert code here...
    NSLog(@"Hello, World!");
    [pool release];
    return 0;
}
```

Let's replace the "insert code here..." comment with a simple for loop we can follow in the debugger. Here's the new version of `main.m`:

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

    // Adding a for loop for debugging fun...
    int count;
    for ( count=1; count<=3; count++ )
    {
        NSLog(@"Hello, World!");
    }

    [pool release];
    return 0;
}
```

Note that we replaced the `NSLog()` call in the original code with a loop that calls `NSLog()` 3 times. Run your new code. You should see the familiar console window, but this time with 3 lines of "Hello, World!" `NSLog()` is a function you can use to get output to the console window and the square brackets are mechanisms you use to send messages to objects designed to receive those messages. In the code above, we send an "alloc" message to the `NSAutoreleasePool` object, then send the resulting object an "init" message. When we are done, we send the object a "release" message. Not to worry, we'll start digging into this syntax next month.

For now, let's take this new code for a spin in the debugger. Close your Run and Build windows (not



Fetch

Best in show.

X

Fetchsoftworks.com

Version 4.0.2 now available.

necessary, just doing this to avoid confusion). In your Project window, note the blank column just to the left of your source code. This column holds your breakpoints and tell the debugger when to stop and wait for your input.

Click just to the left of the for loop to create a breakpoint there. **Figure 5** shows the breakpoint icon that appears.



Figure 5. Click in the source code to create a Breakpoint.

Now let's run the debugger. Click on the hammer/spray can combo icon in the upper-left of the Project window. The debugger window will appear (**Figure 6**), the program will start running, and the debugger will stop at our breakpoint, immediately *before* executing the for loop. Notice the pink highlight bar that highlights the line of code that is *about* to get executed.

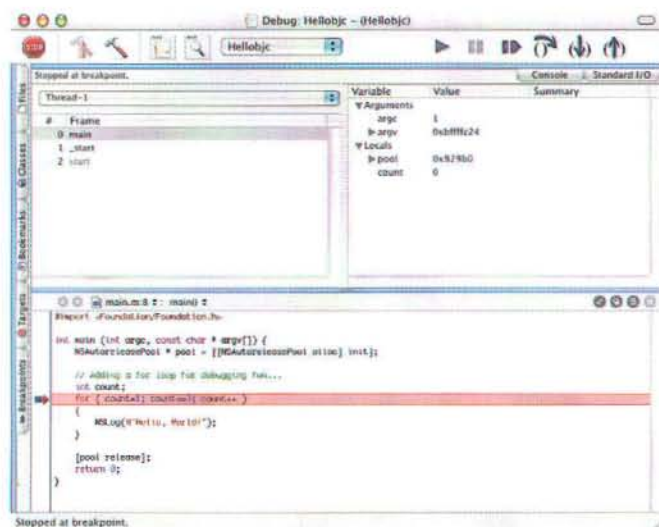


Figure 6. The debugger, stopped just before executing your for loop.

Now step through the execution of your program using the controls that appear in the upper right corner of the debugger window. The triangle restarts execution from the beginning of your program. The second icon, pauses execution, just as if the program had hit a breakpoint. The next icon resumes execution until the next breakpoint or until your program exits. The fourth

icon is one you'll use a lot. Its tool-tip says "Step over method or function call." Basically, it tells your program to keep executing to the next line of code in the current source file. If the current line is a function call, it completely executes the function call, stopping before the next line of code after the function call.

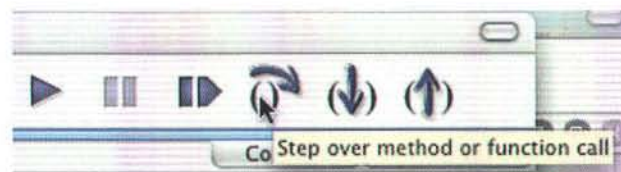


Figure 7. Click "Step over method or function call" to go to the next line of code.

The next icon actually steps into the function, stopping at the first line of code within the function itself. The last icon finishes execution of the current function, stopping at the next line of code within the *calling* function. Add a function to the code and give these controls a try.

As you might expect, the fields that show the variable values update as you step through the program. Notice that the variable `count` starts with a value of 0 and then increments each time through the loop. You can add breakpoints during execution by clicking next to a line of code in the breakpoint column. Want to get rid of a breakpoint? Click and drag to the left or right. Move a breakpoint by dragging up or down.

To view your Standard Library output, click on one of the two tabs towards the upper right corner of the debugger window. The Console pane is a log of important debugger events interleaved with your standard i/o. The Standard I/O window is pure, listing only the i/o itself, each tagged with a time-stamp and a code you can use to link a statement to a specific execution of the program. Each time you restart your program, the code changes, and all i/o from that run will have the same code.

Want to change a variable? Piece of cake. You can double-click on a variable's value and, when the edit field appears, type in a new one. You can also double click on a variable name and a new window will appear, letting you track that variable separately. Cool!

TILL NEXT MONTH...

Spend some time playing with the debugger. Create some code (go back to a Standard Library tool if you are uncomfortable mucking with Objective C), add some functions, play, play, play. The debugger is an incredibly important tool and you should experiment with it until you feel comfortable using it.

See you next time!

This only looks easy...



...this really is easy:

WIBU-KEY Software Protection

■ Software goes online

Electronic Software Distribution – safely protected by WIBU-KEY.

■ Pay-Per-Use

Usage dependent accounting.

■ License Management

You can easily and flexibly create and manage network licenses.

■ Mac OS 9 & X

WIBU-KEY supports Mac OS, Windows and heterogeneous networks.

Test the WIBU-KEY Protection Kit
free and decide for yourself!

1-800-986-6578

sales@griftech.com



The Key is in Your Hands!

**WIBU
SYSTEMS**

WIBU-SYSTEMS AG
76137 Karlsruhe, Germany
WIBU-SYSTEMS USA, Inc.
Seattle, WA 98101
email: info@wibu.com

www.griftech.com
www.wibu.com

Test Kits also available at:

Belgium wibu@impakt.be, Denmark lean@danbit.dk, Finland finbyte@finbyte.com, France info@neot.fr, Hungary info@mrsoft.hu, Japan info@suncaria.co.jp, Jordan, Lebanon starsoft@cyberia.net.lb, Korea dhkimm@wibu.co.kr, Luxembourg wibu@impakt.be, Netherlands wibu@impakt.be, Portugal dubit@dubit.pt, Syria starsoft@cyberia.net.lb, Thailand preecha@dptf-th.com, United Kingdom info@codework.com, USA sales@griftech.com

By Rich Morin

Still More Perl

Munging Mail and Media...

Perl's "whipitupitude" is legendary. This column looks at a couple of small scripts I've recently been "whipping up", showing how Perl can work in and around more formal OSX tools. One script, `fmfm`, Finds Monster Mail Files; I use it to keep track of mailing list (and other) mail files which may be getting out of hand. The other script, `cfwc.d`, is a daemon (background process) which helps me operate an experimental webcam.

FINDING MONSTER MAIL FILES

I'm on quite a few mailing lists and I don't always get to the associated mailboxes regularly to keep them under control. I'm also trying to track the efficacy of my spam filtering system (based on SpamAssassin and Eudora), which drops suspected spam into one of several mailboxes, depending on its numeric spam rating, etc. I have written a short script which helps me keep on top of these issues.

The mainline code, below, is quite simple. Using `finddepth`, from the `File::Find` module (available on the CPAN; cpan.perl.org), it performs a depth-first examination of my email folder. The `callback` function, `wanted`, is invoked for each node (e.g., file, directory) in the tree. Using the lists produced by this traversal, the remaining code prints out the results for spam and miscellaneous email, sorting each list in a case-insensitive manner.

```
#!/usr/bin/env perl
#
# fmfm - find monster mail files
#
# Written by Rich Morin, CFCL, 2002.11

use File::Find;

$monster = 2000000;

{
    $eu = '/Users/rdm/Mail/Eudora Folder';
    finddepth(\&wanted, "$eu/Mail Folder");

    for $line (sort {lc($a) cmp lc($b)} (@spam)) {
        print $line;
    }
    print "\n";

    for $line (sort {lc($a) cmp lc($b)} (@misc)) {
```

```
        print $line;
    }
}
```

The tricky parts of this script, such as they are, lie in the "wanted" callback function. As it traverses the tree, `finddepth` changes the "current directory" and sets `$_` to the relative name of the node. This makes it easy to skip over items that aren't files and Eudora's "table of contents" (`*.toc`) files.

```
sub wanted {
    return unless (-f $_);
    return if ($_ =~ m|\.toc$|);
```

For the next part, however, we need the "full path name" of the node. Getting this from a handy helper method, we can strip off the first part of the path and test the remainder in assorted ways. Perl's regular expressions are very useful for this sort of name handling.

```
$path = $File::Find::name;
$path =~ s|^.*Eudora Folder/Mail Folder/||;
return if ($path =~ m|_Inactive/ Save/|);
```

After picking up the size of the file (in bytes), the script opens each mailbox in the "spam" area and counts the number of "From: headers" (i.e., messages). Eudora uses carriage returns (rather than the conventional BSD newlines) for line termination, but setting Perl's `$/` (input record separator) variable handles that quite easily. The strings containing the formatted output are pushed into a list, for use by the mainline code.

```
$size = -s $_;

if ($path =~ m|!Spam|) {
    open(MBOX, $_) or die "can't open mailbox($_)";
    $/ = "\r";
    $fent = 0;
    while (defined($line = <MBOX>)) {
        $fent++ if ($line =~ m|^From:|);
    }
    close(MBOX);

    push(@spam, sprintf("%-35s %9d %4d\n",
        $path, $size, $fent));
    return;
}
```

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.

The code for miscellaneous mailboxes is comparatively simple. After ensuring that the mailbox is large enough to qualify as a "monster", it formats and saves the output lines. Perl's "x" operator comes in handy for creating a "quick and dirty" histogram.

```
return if ($size < $monster);

$size = int($size/$monster);
push(@misc, sprintf("%-35s %9d %s\n",
    $path, $size, '*' x $size));
}
```

This sort of "personalized" script is quite common in BSD circles. Clearly, it isn't suitable for use by others, as is, but it is short and simple enough that it can easily be customized to meet the needs of different users. Here is some sample output, from my own system:

```
!Spam/?? Junk (Eudora)          9041      5
!Spam/?? Junk (SA 1)           39192     6
!Spam/?? Junk (SA 2)           11467     2
!Spam/?? Junk (SA 3)           420538    60
```

```
_Lists/DocBook                  3231686   *
_Lists/FreeBSD/FreeBSD-Ports     6431902   ***
_Lists/FreeBSD/FreeBSD-Questions 2666962   *
```

A WEBCAM DAEMON

I recently started playing with an iBOT, a FireWire-based camera made by Orange Micro

(www.orangemicro.com). My initial goal was to create a simple "security camera" app that would display a set of recent images on a web page.

After downloading the OSX driver for the iBOT, I started looking around for image capture software. One package, EvoCam (www.evological.com), captures images, based on elapsed time and/or software-based motion detection. It can also upload the image files (via FTP) to a web server and/or save numbered copies on the local disk.

Unfortunately, this wasn't exactly what I wanted. The FTP upload feature simply refreshed the same file; turning this into a time history would be tricky. The numbered image files would do, however, if I could get them over to the web server. All told, it was a good start on what I wanted. All I needed to do was create a little plumbing...

The first part of the plumbing had to do with getting the files from my desktop Mac onto the (FreeBSD-based) local web server. FreeBSD provides NFS, but getting OSX to mount the provided volumes can be quite a trial. Fortunately, Marcel Bresink's NFS Manager (www.bresink.de/osx/NFSManager.html) eases the pain considerably.

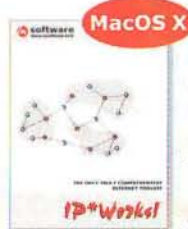
Once I got the files sifting into a directory on the web server, I merely had to rename them (for convenience) and build up a web page to display a selected subset. The following script, while still a "work in progress", accomplishes these tasks quite handily.

```
#!/usr/bin/env perl
#
# cfwc.d - Canta Forda WebCam Daemon
#
# Written by Rich Morin, CFCL, 2002.11

$imgs = '/.../iBOT'; # adjust to taste...
```

THE BEST SELLING INTERNET COMPONENT
SUITE FOR WINDOWS, JAVA, LINUX,
SOLARIS, AND MORE...

NOW RUNNING ON YOUR MAC!



IP*Works! The Only Truly Comprehensive Internet Toolkit

- More Than 30 Components Cover All Major Internet Protocols
- Follows Exact RFC Specifications
- Market-Tested For Over 7 Years
- In Use By Almost All Fortune 500s
- Royalty-Free Pricing

IP*Works! for Mac OS X (10.0/10.1/10.2) gives you access to a host of new capabilities that will connect your applications to the Internet with unprecedented ease of use, power, and flexibility! You will be able to easily and quickly:

- program the web: HTTP, WebForm, WebUpload
- call web services: SOAP, XMLp
- transfer files: FTP, TFTP
- send email: SMTP, FileMailer, HTMLMailer
- receive email: POP, IMAP
- read/post news: NNTP
- encode/decode: MIME, NetCode
- access directories: LDAP
- manage networks: SNMP, Whois, Ping, TraceRoute
- build clients and servers: IPPort, IPDaemon
- build packet applications: UDPPort, Multicast
- remote access: Telnet, Rexec, Rshell, RCP

...and a whole lot more! - download your free trial today from www.nsoftware.com!

IP*Works! for
Mac OS X is currently
available as a C/C++
Library (OS X Framework),
Objective-C Classes for
Cocoa and RealBasic
plugins coming soon at
www.nsoftware.com

STAY TUNED!


```
$html = '/.../cfwc'; # adjust to taste...
```

```
{
  for (;;) {
```

As mentioned above, EvoCam generates a unique name (e.g., 123456789.jpg) for each image file. In writing these to the NFS-mounted FreeBSD machine, OSX also generates a companion file (e.g., _123456789.jpg) for the resource fork. The code below creates a new name for the image file, based on the file's modification time, and discards the companion file.

```
# Clean out incoming directory.
```

```
opendir(IN, "$imgs/incoming")
  or die "can't open $imgs/incoming";
@in = grep(!/^\.\/, readdir(IN));
chomp(@in);
closedir(IN);

for $in (sort(@in)) {
  @stat = stat("$imgs/incoming/$in");
  $mtime = $stat[9];

  ($sec, $min, $hour, $mday, $mon, $year,
   $yday, $yday, $isdst) = localtime($mtime);

  $out = sprintf("%d.%02d%02d.%02d%02d%02d.jpg",
    $year+1900, $mon+1, $mday, $hour, $min, $sec);

  rename("$imgs/incoming/$in",
    "$imgs/i.queue/$out");
  unlink("$imgs/incoming/_.$in");
}
```

Perl's approach to reading directories is rather messy, but it isn't all that difficult. The code below gets a list of filenames, discarding any that don't match the desired format, and sorts them. Because the names were crafted with this in mind, the list is now in chronological order.

```
# Get list of images to display.
```

```
opendir(IN, "$imgs/i.queue")
  or die "can't open $imgs/i.queue";
@in = sort(grep(/^(\d{4})\.\d{4}\.\d{6}\.jpg$/,
  readdir(IN)));
chomp(@in);
closedir(IN);
```

Using Perl's "slice" syntax, we grab the last (i.e., most recent) nine file names.

```
@show = @in[-9 .. -1];
```

Now we start generating a web page. The META tag tells the user's browser to refresh the page every 15 seconds. I am rather compulsive about formatting the HTML; the web browser doesn't care, but it sure makes debugging less painful for humans!

```
# Make up a new web page.
```

```
open(OUT, ">$html/index.temp")
  or die "can't open index.temp";

print OUT <<EOT;
<HTML>
<HEAD>
  <META HTTP-EQUIV="Refresh" content="15">
  <TITLE>Canta Forda WebCam</TITLE>
</HEAD>

<BODY>
  <TABLE>
```

BOT

The code below generates a 3x3 table of images, each followed by a centered label. I could have used the file names (e.g., 2002.1129.2039.jpg) as labels, but that would have been a bit ugly. Why not parse the names and reformat the values into a more readable format?

Note the multi-line regular expression that is used to break up the file name. When REs get long and complex, breaking them up in this manner can make them much easier to follow.

```
$cnt = 0;
for ($i=0; $i<9; $i+=3) {
  print OUT "      <TR>\n";
  for ($j=0; $j<3; $j++) {
    print OUT "          <TD>\n";

    $k = $i + $j;
    $tmp1 = $show[$k];
    $tmp1 =~
      m|^(\d{4})\.\d{4}\.\d{6}\.jpg$|x; # (YYYY).
      (\d\d)(\d\d)\. # (MM)(DD).
      (\d\d)(\d\d)(\d\d)\. # (HH)(MM)(SS).
      jpg|x; # jpg
    $tmp2 = sprintf("%s/%s/%s at %s:%s:%s",
      $1, $2, $3, $4, $5, $6);

    print OUT "          <CENTER>\n";
    print OUT "          " .
      "<IMG SRC=\"$iq/$tmp1\"><BR>\n";
    print OUT "          $tmp2\n";
    print OUT "          </CENTER>\n";
    $cnt++;

    print OUT "          </TD>\n";
  }
  print OUT "      </TR>\n";
}
```

Finally, we push out the last of the HTML, close the file and (Oh, yes!) move it into place for Apache to find. Then, after a second's repose, we go back up and do the whole exercise again.

```
print OUT <<EOT;
</TABLE>
</BODY>
</HTML>
EOT

close(OUT);
rename("$html/index.temp",
  "$html/index.html");
sleep(1);
}
```

LESSONS LEARNED

As we all know, the Mac and BSD universes aren't a perfect fit. Perl is a very good "glue language", however, allowing us to deal smoothly with issues such as line termination, extra (e.g., resource fork) files, etc.

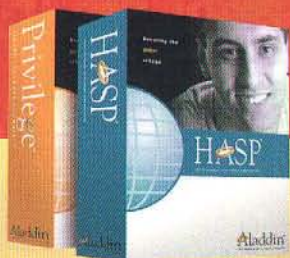
Similarly, there are a wealth of useful apps which can perform small tasks, fill in gaps between operating systems, and generally make our lives easier. If a \$20 shareware package can save me hours of frustration, the purchase decision is a no-brainer.

Unfortunately, some issues are still difficult to resolve. For instance, although it's easy to scan a Eudora mail file for header lines, editing Eudora mailboxes would be far trickier. Aside from file locking problems, there is the small issue of the (binary, undocumented) format of the TOC files. In short, choose your challenges carefully...



CHICKENS SHOULD ROAM FREE. YOUR SOFTWARE SHOULDN'T.

Try our
Piracy Calculator.
See how much revenue may
have already flown the
coop at hasp.com/cal.



HASP® & Privilege®

Don't let "free roaming" software result in lost revenues — take advantage of the superior software licensing solutions offered by Aladdin Knowledge Systems. Did you know some businesses lose almost 50% of their revenue due to software piracy? That's why over 25,000 companies trust their software

licensing to Aladdin. Whether you prefer the proven reliability of HASP® dongles (USB or parallel port) or the state-of-the-art downloading of Privilege™, Aladdin products have a 99.97% success rate. Call us at 1-800-562-2543 to prevent your profits from flying the coop.

Aladdin®
SECURING THE GLOBAL VILLAGE
eAladdin.com

By Andrew S. Downs

Dock Tile Imaging using JDirect

A pure Java approach to changing a Java application's dock tile at runtime

OVERVIEW

A previous article ("Dock Tile Imaging", MacTech June 2002) discussed using the Java Native Interface technology to dynamically change the dock tile for a Java application. This article illustrates an alternate approach: the use of Apple's JDirect technology to make the native API calls directly from Java without writing any C code. This eliminates the need for a shared library and the scaffolding to call functions in that library.

The application discussed in this article periodically fetches a new image from NASA's Kennedy Space Center video feed web page. The source page contains links to 15 channels (images). The application parses the source page to find the individual image links, then sequentially downloads each image. In addition to displaying each retrieved image, the application paints a countdown progress bar in the tile between fetches. This progress bar painting uses the QuickDraw API, called via JDirect.

APPLICATION STARTUP

The JDirectDockTiler class contains main(), the entry point for the Java application. main() is used simply to instantiate the DockTiler class and invoke an instance method. DockTiler creates an instance of UrlDownloader, which fetches an image from a URL. UrlDownloader invokes a method in DockTiler, passing it the image bytes. Some of the code in this class is reused from the June 2002 article, but is not discussed again. Download the accompanying source files for this article, refer to the previous article, or send me email if you need more info.

The import statements include two Macintosh Runtime for Java (MRJ) packages, one of which is JDirect. The other package, macos.frameworks, contains declarations for the Carbon and ApplicationServices interfaces.

```
import com.apple.mrj.jdirect.*;
import com.apple.mrj.macos.frameworks.*;
```

```
public class JDirectDockTiler {
    public static void main (String args[]) {
        DockTiler dock = new DockTiler();
        dock.run();
    }
}
```

```
class DockTiler {
```

If we have trouble loading an image, the values of its width and height will remain -1. See loadImage().

```
int mWidth = -1, mHeight = -1;
```

The pixels are stored as a 1-dimensional array of integers. Three bytes of the int contain the RGB values, while the fourth byte contains the alpha (transparency) value.

```
int mPixels[];

public DockTiler() {
}
```

Instantiate a class that handles periodic image fetching and display.

```
public void run() {
    UrlDownloader u = new UrlDownloader( this );

    u.setUrl(
        "http://spaceflight.nasa.gov/realdata/ksclive/index.html"
    );
```

This substring will be located in the fetched page. This is a potential problem if the source page content changes, since then our substring search may stop working. This is a good candidate for a property in an external file.

```
u.setSearchString( "http://science.ksc.nasa.gov" );
```

The number of iterations indicates how many times the substring occurs in the page. This is another potential trouble spot that could result in something breaking.

```
u.setIterations( 15 );
```

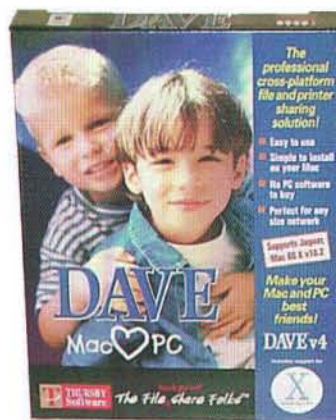
Andrew has been a Java fan since his first encounter with the language at the WebEdge III conference in 1996. You can reach him at andrew@downs.ws.

Stop by
our booth #2111
at Macworld Expo
San Francisco!

With DAVE[®] your Windows network will welcome Mac users!



*The door is wide open for **sharing files and printers** between Macs and PCs. DAVE offers maximum cross-platform performance and productivity because DAVE:*



- Allows the Mac to be a full partner in a Windows network.
- Supports OS 8.6-9.2x as well as OS X 10.1.5-10.2x.
- Installs only on the Mac...nothing is added to the PC or the server.
- Supports Microsoft standard NTFS file format.
- Provides three options for Security.
- Offers a FREE fully functional demo at www.thursby.com/evaluations.

Download
DAVE[®]
TODAY.

www.thursby.com



Fortunately, the Thread sleep interval is entirely within our control. The value 5000 shown here indicates the Thread should sleep for 5 seconds between invocations.

```
u.setSleep( 5000 );
u.start();
}

static Frame f = null;
static PictureFrame pf = null;
```

This version of loadImage() differs from the JNI version in that it uses an array of bytes to create an image. Also, the grabPixels() completion checking is slightly more complicated in order to prevent partial or blank images from occurring.

```
protected void loadImage( byte [] array ) {
    Image image =
        Toolkit.getDefaultToolkit().createImage( array );

    if ( f != null && pf != null )
        f.remove( pf );

    if ( f == null )
        f = new Frame( "Image" );

    // Offscreen windows move onscreen when you change the dock settings.
    // Keeping it onscreen allows you to view the original image as well
    // as the dock tile.
    f.setBounds( 50, 50, 200, 200 ); // onscreen
    // f.setBounds( -250, -250, 200, 200 ); // offscreen
    f.setLayout( new BorderLayout() );

    pf = new PictureFrame( image );

    f.add( "Center", pf );
    pf.setSize( 128, 128 );

    f.pack();
    f.setVisible( true );
    f.repaint();

    pf.repaint();

    int width = image.getWidth( pf );
    int height = image.getHeight( pf );

    if ( width < 0 || height < 0 )
        return;

    mPixels = new int[ width * height ];

    for ( int i = 0; i < width * height; i++ )
        mPixels[ i ] = 0;

    PixelGrabber pg = new PixelGrabber( image, 0, 0, width,
        height, mPixels, 0, width );

    try {
        pg.grabPixels();

        while ( ( pg.getStatus() & ImageObserver.ALLBITS )
            == 0 ) {
            if ( ( pg.getStatus() & ImageObserver.ABORT )
                != 0 )
                return;

            Thread.sleep( 3000 );
        }

        mWidth = width;
        mHeight = height;
    }
    catch ( InterruptedException e ) {
        return;
    }
}
```

```
if ( mWidth > 0 && mHeight > 0 )
    ImageUtils.setDockTile( mPixels, mWidth, mHeight );
}
```

RETRIEVING IMAGES

There are undoubtedly third party, or perhaps Sun-provided, classes that can fetch and parse HTML pages, but the one discussed here contains the necessary functionality for this app.

The methods in this class:

- fetch a page at a given URL
- locate image links with a page
- fetch an image at a specified URL

```
public class UrlDownloader {
```

The following attributes have corresponding accessors (not shown):

```
String mUrl = "";
int mSleep = 0;
String searchString = "";
static int mCount = 0;
```

This corresponds to the number of iterations.

```
static int max = 1;
```

This attribute is set from the constructor, but could also be done with an accessor:

```
DockTiler tiler;

UrlDownloader( DockTiler dt ) {
    tiler = dt;
}

public void start() {
    downloadPage();
}
```

Timer is the actual Thread, and is discussed further down.

```
Timer t = new Timer( this, mSleep );
t.run();
}
```

This buffer holds the raw downloaded page content.

```
StringBuffer sb = new StringBuffer();
```

This method uses a URL string to open a connection and download the content of that page.

```
protected void downloadPage() {
    try {
        URL url = new URL( mUrl );
        URLConnection conn = url.openConnection();
        int length = conn.getContentLength();

        BufferedInputStream is =
            new BufferedInputStream( url.openStream() );
    }
}
```



```
int total = 0;
int numRead = 1;
```

Rather than read the page as one huge chunk of data, read 1k pieces and append them to the buffer.

```
while ( numRead > 0 && total < length ) {
    byte array[] = new byte[ 1024 ];
    numRead = is.read( array, 0, 1024 );
    total += numRead;

    String s = new String( array );
    sb.append( s );
}
catch ( MalformedURLException e ) {
    System.out.println( e );
}
catch ( IOException e ) {
    System.out.println( e );
}
}
```

This method gets called by the timer Thread. It dispatches to methods that locate the next occurrence of the search string on the page, fetch the appropriate image, then call back to DockTiler to display to image.

```
protected void fetch() {
    String s = parseUrl();
    byte array[] = fetchImage( s );
    tiler.loadImage( array );
}
```

```
protected String parseUrl() {
    String url = new String();
```

Update the current image number. Reset after reaching the total number of image links on the page.

```
mCount++;
if ( mCount > max )
    mCount = 1;

int occurrence = 0;
int last = 0;

String page = sb.toString();

int index = 0, prevIndex = 0;
```

Find the beginning of the next image link on the page.

```
while ( occurrence < mCount ) {
    index = page.indexOf( searchString, prevIndex );
    prevIndex = index + 1;
    occurrence++;
}
```

Find the end of the link: a quote character.

```
if ( index > 0 )
    url = page.substring( index,
        page.indexOf( "\"", index + 1 ) );

return url;
}
```

```
protected byte[] fetchImage( String imageUrl ) {
```

Return a 1-byte array in lieu of an error code in case of failure.

```
byte array[] = new byte[ 1 ];

if ( !imageUrl.startsWith( "http" ) ) {
    System.out.println( "Bad URL: " + imageUrl );
    return array;
}

try {
```

This is similar to how we fetched the page contents above, except that readFully() eliminates the loop.

```
URL url = new URL( imageUrl );
URLConnection conn = url.openConnection();
int length = conn.getContentLength();

array = new byte[ length ];

DataInputStream is =
    new DataInputStream( url.openStream() );

int total = 0;
int numRead = 1;

is.readFully( array );
}
catch ( MalformedURLException e ) {
    System.out.println( "Bad URL: " + imageUrl );
```



IMPACT YOUR BOTTOM LINE


Core Competencies

- Mac OS X application development
- Cocoa application, Carbon porting
- Mac OS product maintenance
- Windows/ Mac OS/ Unix porting
- Cross-platform development
- WebObjects development

Service Offerings

- On-Site staff augmentation
- Off-Site project development
- Offshore project development
- User training

For more info
<http://www.avestacs.com>
 Email: mithu@avestacs.com
 Tel: 201-369-9400 Ext. 237



Avesta Computer Services, Ltd.


```

    array = new byte[ 1 ];
}
catch ( IOException e ) {
    System.out.println( e );
}
catch ( NegativeArraySizeException e ) {
    System.out.println( "Bad array size: " + e );
    array = new byte[ 1 ];
}

return array;
}
}

```

This simple timer class calls the `fetch()` dispatch function, then sleeps for 1/100th of the specified Thread sleep time. In between, it updates the progress bar at the bottom of the dock tile image. Upon reaching 100% of the sleep interval, start another fetch cycle. To remove the progress bar updates, uncomment the specified `Thread.sleep()` call and remove the `while` loop.

```

class Timer implements Runnable {
    UrlDownloader mUrlDownloader = null;
    int mSleep = 5000;

    Timer( UrlDownloader u, int sleep ) {
        mUrlDownloader = u;
        mSleep = sleep;
    }

    public void run() {
        while ( true ) {
            try {
                mUrlDownloader.fetch();
                // Replace following block with this line to avoid drawing of progress bar.
                // Thread.sleep( mSleep );

                int total = 0;
                int i = 0;
                float interval = mSleep / 100;

                while ( i <= 100 ) {
                    ImageUtils.updateProgressBar( ( int )i );
                    Thread.sleep( ( int )interval );
                    i++;
                }
            }
            catch ( InterruptedException e ) {
            }
        }
    }
}

```

CALLING QUARTZ

The `ImageUtils` class provides Java access to various native Mac OS API calls, such as the Quartz 2D (Core Graphics) library. The imports before the class definition include the `JDirect` package (containing the `Linker` class that is used to register the native methods) and the `macos.frameworks` package (which defines the `Carbon` and `ApplicationServices` interfaces):

```

import com.apple.mrj.jdirect.*;
import com.apple.mrj.macos.frameworks.*;

public class ImageUtils implements Carbon,
    ApplicationServices {

```

The `Linker` constructor call registers the native methods declared in this class as JNI methods:

```

private static final Object linkage =
    new Linker( ImageUtils.class );

```

The Mac OS calls must be declared explicitly. Argument names are not important here. I used letters as placeholders, but if you intend to reuse such a class it may make sense to provide more descriptive names and possibly comments or documentation so you don't have to revisit the API documentation each time. These two Core Graphics declarations illustrate native methods without and with arguments:

```

public static native int
    BeginCGContextForApplicationDockTile();

public static native int CGDataProviderCreateWithData(
    int a, int [] b, int c, int d );

```

The non-native method `setDockTile()` draws an image using the Core Graphics API. The arguments include an array of integers (4-bytes) representing the RGBA values for each pixel, and the image width and height.

```

static protected void setDockTile( int [] imagePixels,
    int width, int height ) {

```

This value defines the number of bytes in each pixel:

```

int kNumComponents = 4;

```

This is somewhat annoying, having to redefine the constant values already defined by the API:

```

final int kCGImageAlphaFirst = 4;
final int kCGRenderingIntentDefault = 0;

```

If you look at the API docs, you will notice that many of the values used or returned by the native calls are pointers. In `JDirect` (as in `JNI`), pointers map to the `int` data type. This call gets the graphics context for the current application's dock tile:

```

int theContext =
    BeginCGContextForApplicationDockTile();

if ( theContext != 0 ) {

```

The number of bytes comprising one row of the image may be calculated using the number of pixels per row and the number of bytes per pixel:

```

int bytesPerRow = width * kNumComponents;

```

Create a data source using the pixel array passed in as the image source. Then create an RGB color space object.

```

int theProvider = CGDataProviderCreateWithData( 0,
    imagePixels, ( bytesPerRow * height ), 0 );

```


Multiple formats. Multiple platforms. Complex installers.

Aladdin solves the compression and installation puzzle.

**Trying to figure out how to handle multiple
compression formats and platforms?**

The StuffIt Engine solves the compression puzzle.



Aladdin's StuffIt Engine SDK:

- Adds value to your application by integrating powerful compression and encryption.
- Is the only tool that supports the StuffIt file format.
- Provides a single API that supports over 20 compression and encoding formats common on Macintosh, Windows, and Unix.
- Makes self-extracting archives for either Macintosh or Windows.
- Available for Macintosh, Windows, Linux, or Solaris.

Licenses start as low
as \$99/year

To learn more, visit:
www.stuffit.com/sdk/

StuffIt Engine SDK™ The power of StuffIt in your software.



**Looking for the easiest and fastest
way to build an installer?**

StuffIt InstallerMaker completes your puzzle.

It's not enough just to write solid code anymore. You still have to write an installer for your users. StuffIt InstallerMaker makes it simple and effective.

- StuffIt InstallerMaker gives you all the tools you need to install, uninstall, resource-compress or update your software in one complete, easy-to-use package.
- Add marketing muscle to your installers by customizing your electronic registration form to include surveys and special offers.
- Make demoware in minutes. Create Macintosh OS X and Macintosh Classic compatible installers with StuffIt InstallerMaker.

Prices start at \$250

To learn more, visit:
[www.stuffit.com/
installermaker/](http://www.stuffit.com/installermaker/)

StuffIt InstallerMaker™ The complete installation solution.™



www.stuffit.com
(831) 761-6200

© 2002 Aladdin Systems, Inc. StuffIt, StuffIt InstallerMaker, and StuffIt Engine SDK are trademarks of Aladdin Systems, Inc. The Aladdin logo is a registered trademark. All other products are trademarks or registered trademarks of their respective holders. All Rights Reserved.


```
int theColorspace = CGColorSpaceCreateDeviceRGB();
```

Now create the image. This is similar to creating a `Pixmap`. The symbol `kCGImageAlphaFirst` specifies that in the pixel data, the alpha channel (transparency) value is in the first byte of each int. More information can be found at: http://developer.apple.com/techpubs/macosx/CoreTechnologies/graphics/Quartz2D/Quartz_2D_Ref/index.html.

```
int theImage = CGImageCreate( width, height, 8, 32,
    bytesPerRow, theColorspace,
    kCGImageAlphaFirst, theProvider, 0, 0,
    kCGRenderingIntentDefault );
```

Cleanup by explicitly releasing pointers to native objects. We could have left this until the end and cleaned up all resources at the same time.

```
CGDataProviderRelease( theProvider );
CGColorSpaceRelease( theColorspace );
```

Use the newly created image as the tile image:

```
int theError =
    SetApplicationDockTileImage( theImage );
```

Flush to ensure the pixels get written:

```
CGContextFlush( theContext );
```

Finally, free the native image and the drawing context:

```
CGImageRelease( theImage );
EndCGContextForApplicationDockTile( theContext );
}
```

CALLING QUICKDRAW

This portion of the `ImageUtils` class is used to paint a progress bar over bottom of the image in the dock tile while the application sleeps prior to fetching a new image.

The `QuickDraw` Toolbox declarations look the same as the `Core Graphics` method declarations just discussed. However, since `QuickDraw` uses non-opaque data structures in various places this set of declarations includes as arguments, for example, an array of shorts to represent a native `Rect`.

```
public static native int
    BeginQDContextForApplicationDockTile();
public static native int GetPortBounds( int port,
    short [] rect );
public static native void SetRect( short [] r,
    short left, short top, short right, short bottom );
```

Several color constants defined in the `Toolbox` must be defined here as well before they can be used:

```
static final int blackColor = 33;
static final int redColor = 205;
static final int greenColor = 341;
```

The progress bar is a 10-pixel tall rectangle superimposed on the lower-edge of the dock tile. The background of the progress bar is red, with a black frame. Each update fills a larger percentage of the bar with green.

```
static protected void updateProgressBar(
    int currPercent ) {
```

First obtain a reference to the tile drawing surface:

```
int thePort = BeginQDContextForApplicationDockTile();
if ( thePort != 0 ) {
```

Next, create an instance of a `Rect` class (discussed in the next section). That rectangle will hold the boundaries of the drawing surface.

```
Rect theRect = new Rect();
GetPortBounds( thePort, theRect.getArray() );
```

The initial call simply frames the empty red rectangle:

```
if ( currPercent == 0 ) {
    SetRect( theRect.getArray(), theRect.getLeft(),
        ( short )( theRect.getBottom() - ( short )10 ),
        theRect.getRight(), theRect.getBottom() );
    ForeColor( redColor );
    PaintRect( theRect.getArray() );
    ForeColor( blackColor );
    FrameRect( theRect.getArray() );
}
```

The right edge of the bar must be calculated using the current percent complete and the width of the bar in pixels. The right edge is treated as a float because of the fractions encountered during the calculation.

```
float right = 0;
if ( currPercent >= 100 )
    right = ( float )theRect.getRight();
else
    right = ( ( ( float )theRect.getRight() -
        ( float )theRect.getLeft() ) /
        ( float )100 ) * ( float )currPercent;
```

Fill the "percent complete" area with green. The right edge value calculated above gets cast to a short data type and used to define the rectangle to be painted:

```
ForeColor( greenColor );
SetRect( theRect.getArray(),
    ( short )( theRect.getLeft() + 1 ),
    ( short )( theRect.getBottom() - 9 ),
    ( short )right,
    ( short )( theRect.getBottom() - 1 ) );
PaintRect( theRect.getArray() );
```

Flush the bits to the screen and cleanup:

```
QDFlushPortBuffer( thePort, 0 );
EndQDContextForApplicationDockTile( thePort );
}
```


RECT SUPPORT CLASS

JDirect relies on various Toolbox structures that exist in native Mac OS libraries, but need to be explicitly defined for use with JDirect. Here is one way to define a class that corresponds to the Rect data structure:

```
public class Rect {
    private short [] theRect = new short[ 4 ];
```

The four corners of the Rect are stored in an array whose sequence we arbitrarily define as:

```
theRect[ 0 ] = Left
theRect[ 1 ] = Top
theRect[ 2 ] = Right
theRect[ 3 ] = Bottom
```

The storage assignments are arbitrary because access to the actual values is provided through accessor methods only, such as:

```
public short getLeft() {
    return theRect[ 0 ];
}

public void setLeft( short i ) {
    theRect[ 0 ] = i;
}
```

The constructor initializes the fields. The explicit casts to the short (16-bit) data type convert each 0 value, which defaults to type int (32-bit).

```
public Rect() {
    setLeft( ( short )0 );
    setTop( ( short )0 );
    setRight( ( short )0 );
    setBottom( ( short )0 );
}
```

A more compact though implementation of this class could eliminate the accessor methods and allow direct access to the data values, perhaps as four short values. That exposure may lead to problems should you wish to change the internal data representation. Always try to hide the internals of a class and enforce access through an API.

A more complex get/set combination that hides the internal representation can be implemented using constant values to represent each corner, as in:

```
static final int kLeft = 0, kTop = 1, kRight = 2,
    kBottom = 3;

public short get( short corner ) {
    if ( corner >= kLeft && corner <= kRight )
        return theRect[ corner ];
    else
        // Not good because 0 is a valid value for any corner.
        return 0;
}

public void set( short corner, short i ) {
    if ( corner >= kLeft && corner <= kRight )
        theRect[ corner ] = i;
    // What if the caller passes a bad corner identifier?
}
```

Although the variations just discussed have their merits, the Rect implementation for this project uses explicit get/set methods for each corner of the Rect. The class definition is slightly longer, but more forgiving, than an implementation requiring the caller to specify a numeric value for a corner as an additional argument to both get/set. Also, this implementation does not expose the internal data representation.

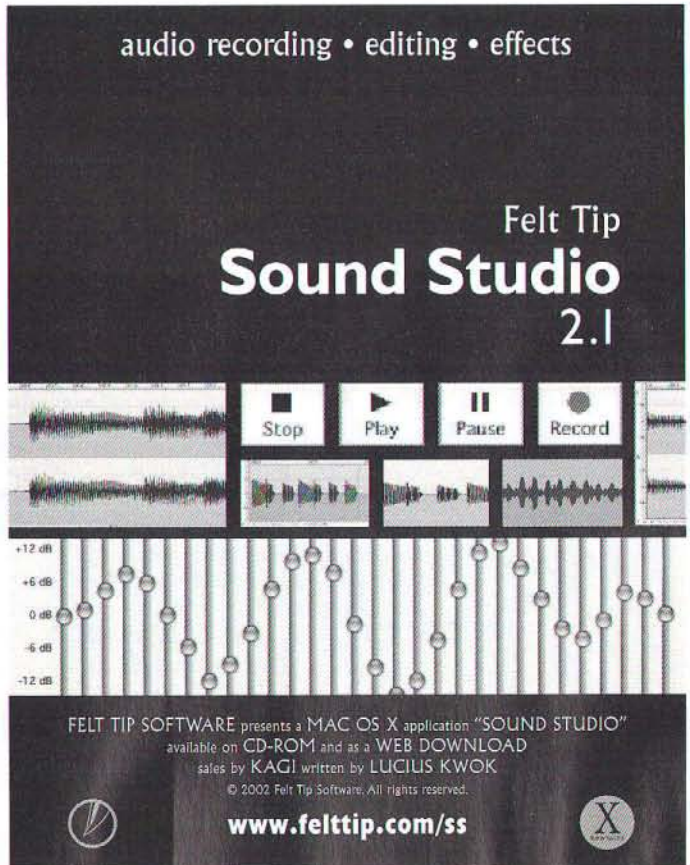
REFERENCES

The book *Early Adopter Mac OS X Java* (Wrox Press, Inc., November 2001) contains an excellent chapter on integrating Java with native Mac OS calls using JDirect. TechNote 2002 (<http://developer.apple.com/technotes/tn/tn2002.html>) also discusses this issue.

Thank you to Eric Albert, Greg Parker, and Nicholas Riley for their assistance and advice while I wrote this program at MacHack.

audio recording • editing • effects

Felt Tip
Sound Studio
2.1



FELT TIP SOFTWARE presents a MAC OS X application "SOUND STUDIO" available on CD-ROM and as a WEB DOWNLOAD sales by KAGI written by LUCIUS KWOK © 2002 Felt Tip Software, All rights reserved.

www.felttip.com/ss

by Tim Monroe

Animal Crackers

Enhancing Cocoa QuickTime Applications

INTRODUCTION

In the previous *QuickTime Toolkit* article ("The Cocoanuts" in *MacTech*, December 2002), we took a look at using Cocoa, Apple's object-oriented application framework, to develop QuickTime applications. We saw how to put together a basic multi-window application that can open, play, and edit QuickTime movies, and we also investigated a few ways to extend that application by subclassing Cocoa's built-in QuickTime classes or by adding additional methods to those classes with categories.

In this article, I want to continue investigating Cocoa and QuickTime. Primarily, I want to tie up a few loose ends that we didn't have time to address in the previous article. For instance, we need to do a little bit more work to get all the items in the Edit and File menus working as expected, and we need to fix a few minor cosmetic problems with the application. Consider this article then to be some fine tuning to get our Cocoa application, MooVeez, to provide all the functionality of our sample application QTShell. Along the way, however, I also want to take time to investigate a few new topics, including the wired action introduced in QuickTime 6 that allows us to set a movie's scale. And, believe it or not, I'm going to slip in a few tweaks to QTShell itself. So don't be surprised if you see some Carbon code along the way.

Before we begin, I should note that in this article (and the previous one), we're relying on the features of QuickTime and Cocoa that shipped with the so-called Jaguar release: Mac OS X version 10.2 or later, and QuickTime 6.0 or later. Earlier versions of the QuickTime Cocoa classes, **NSMovie** and **NSMovieView**, have some noticeable problems in a few key areas. The Jaguar versions of these classes are much more reliable and efficient, so that's what I'll assume we're working with.

DISPLAYABLE PATHS

Let's begin with an easy but important update to our existing MooVeez code. Recall that our application can open a QuickTime

movie in a window, and that this window provides a pop-out drawer listing some basic information about the movie (**Figure 1**).

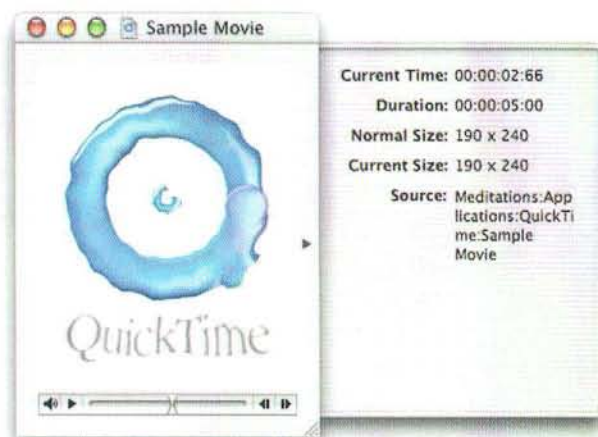


Figure 1: A movie information drawer

The "Source" is the full path of the QuickTime movie file. In the previous article, I offered the **setSource:** method shown in **Listing 1**, which breaks the pathname into its components, tosses out the string "Volumes", and then rebuilds a path using the colon (":") as the path separator.

Listing 1: Displaying the movie's source

```

- (void)setSource: (NSString *)name
{
    NSArray *pathComponents = [name
        componentsSeparatedByString:@"/"];
    NSEnumerator *pathEnumerator = [pathComponents
        objectEnumerator];
    NSString *component = [pathEnumerator nextObject];
    NSMutableString *massagedPath = [NSMutableString string];

    while (component != nil) {
        if ([component length] > 0) &&
            (strcmp([component cString], "Volumes") != 0)) {
            [massagedPath appendString:component];

            component = [pathEnumerator nextObject];
            if (component != nil)
                [massagedPath appendString:@":"];
        } else {
            component = [pathEnumerator nextObject];
        }
    }
}
    
```

Tim Monroe is a member of the QuickTime engineering team. You can contact him at monroe@apple.com. The views expressed here are not necessarily shared by his employer.



BALANCED COLORS

READ & LEARN > BUY SMART

REVIEW > BEST INKJET PRINTERS > IMAGE BY ANTHONY SAINT JAMES

CREATIVE DESIGNERS, WRITERS, MUSICIANS, BUSINESS LEADERS AND OUR TECHNICAL EXPERT TEAM OFFER THEIR OWN PERSONAL INTERPRETATION OF THINGS THAT ONLY THE MACINTOSH SYSTEM CAN DRIVE. FEATURING OVER 240 PAGES OF REVIEWS, INTERVIEWS, PRODUCT NEWS, INSIGHTS, TRENDS AND THE LARGEST MACINTOSH BUYERS' GUIDE. SUBSCRIBE > \$32/1 YEAR (4 ISSUES) OR \$62/2 YEARS (8 ISSUES): WWW.MACDIRECTORY.COM/PAGES/ADVERT.HTML OR SEND CHECK OR MONEY ORDER TO: MACDIRECTORY SUB DEPT. 326 A STREET, 2C, SOUTH BOSTON MA 02110

MacDirectory


```

    }
    [_sourceName setStringValue: massagedPath];
}

```

This isn't quite right, however, since it will toss out a path component named "Volumes" wherever it occurs in the full pathname (not just at the beginning).

I have subsequently learned that there is a Cocoa method that can assist us here; the **NSFileManager** class provides the **componentsToDisplayForPath:** method, which returns an array containing the components of a file's *displayable name* (that is, the name that should be displayed to the user). So if, as above, **name** is the full UNIX pathname of a file, we can get the array of displayable components like this:

```

NSArray *pathComponents = [[NSFileManager defaultManager]
    componentsToDisplayForPath:name];

```

For instance, the full UNIX pathname of the movie file shown in **Figure 1** is:

```
/Volumes/Meditations/Applications/QuickTime/Sample Movie
```

The **componentsToDisplayForPath:** method returns an array whose 4 elements are these:

```

Meditations
Applications
QuickTime
Sample Movie

```

Then we just need to reassemble these components with the appropriate path separator. **Listing 2** shows our revised version of the **setSource:** method.

Listing 2: Displaying the movie's source (revised)

```

- (void)setSource:(NSString *)name                               setSource
{
    NSArray *pathComponents = [[NSFileManager defaultManager]
        componentsToDisplayForPath:name];
    NSEnumerator *pathEnumerator = [pathComponents
        objectEnumerator];
    NSString *component = [pathEnumerator nextObject];
    NSMutableString *displayablePath =
        [NSMutableString string];

    while (component != nil) {
        if ([component length] > 0) {
            [displayablePath appendString:component];

            component = [pathEnumerator nextObject];
            if (component != nil)
                [displayablePath appendString:@" "];
        } else {
            component = [pathEnumerator nextObject];
        }
    }

    [_sourceName setStringValue:displayablePath];
}

```

It's perhaps worth mentioning that Carbon applications can call the Launch Services functions **LSCopyDisplayNameForRef** or **LSCopyDisplayNameForURL** to get displayable path names.

EDIT MENU

There is one outright bug in the Jaguar implementation of **NSMovieView** that rears its head when we try to undo a change to a movie. Do this: open a movie, select some of the movie data using the controller bar, and then cut the selected data. As expected, the Paste menu item is now enabled; the problem is that the Undo menu item is not enabled. (See **Figure 2**.) We *ought* to be able to undo the cut, and **NSMovieView** *should* be handling this automatically for us, but it's not.



Figure 2: The Edit menu of MooVeez

The Cocoa engineers are aware of this problem, and I believe that a fix has been found and will be incorporated into a future version of **NSMovieView**. In the meantime, there appears to be a fairly simple workaround to this problem. The workaround involves subclassing **NSMovieView** so that we can override the **validateMenuItem:** method. **Listing 3** shows the override method.

Listing 3: Enabling and disabling the Undo menu item

```

- (BOOL)validateMenuItem:(NSMenuItem *)item                    validateMenuItem
{
    BOOL isValid = NO;
    SEL action = [item action];

    // handle the Undo menu item
    if (action == @selector(undo:)) {
        MovieController mc = [self movieController];

        if (mc) {
            long flags = 0L;

            MCGetControllerInfo(mc, &flags);
            isValid = flags & mcInfoUndoAvailable;
        }
        else {
            isValid = [super validateMenuItem:item];
        }
    }

    return isValid;
}

```

Here we're just calling **MCGetControllerInfo** and checking the returned set of flags to see whether the **mcInfoUndoAvailable** flag is set. If it is, we enable the Undo menu item; otherwise we disable the menu item. Notice that we pass all other menu items to the superclass (that is, to **NSMovieView**).

Once we've added this code to our subclass of **NSMovieView**, the Undo and Redo menu items are enabled and disabled as expected, as shown in **Figure 3**. In this case, the user has already cut some of the movie data and then undone the cut (that's why the Redo menu item is enabled).

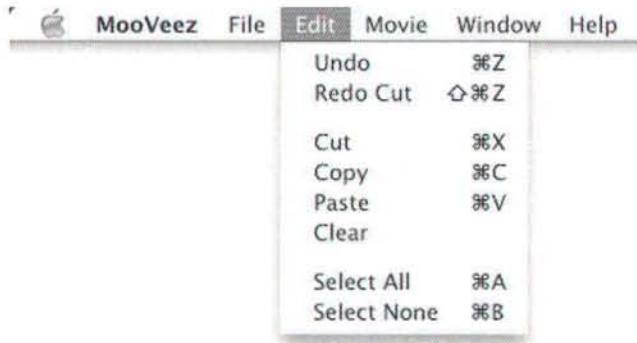


Figure 3: The Edit menu of MooVeez (revised)

In addition, the Undo and Redo menu items appear to function correctly now. Take this new code for a spin: undo some cuts and pastes and see if everything works as expected. My preliminary tests do not reveal any problems, but I'm a bit worried by the following console message that appears the first time the user performs an undo operation:

```
Warning: -[NSMovieView undo:] is obsolete.
```

I recommend employing this workaround only after thoroughly testing it yourself.

FILE MENU

Now that we've got the Edit menu working pretty much as expected, let's take a quick look back at the File menu. In the previous article, we explicitly postponed adding support for creating new, empty movie windows — that is, supporting the New menu item. To disable the New menu item, we added a **validateMenuItem:** method to our application controller class (defined in the file **AppController.m**). And we overrode the **newDocument:** method to act as a no-op. Since we want to add support for creating new empty movies, we need to remove these methods from our application controller class entirely. Once we do that, the default behaviors of the **NSDocument** class will kick in and we'll be able to create new empty documents.

For this to work correctly, however, we need to make a few simple fixes to the code that is called by the **windowControllerDidLoadNib:** method. Hitherto we have assumed that the user had opened a movie file by selecting the Open or "Open Recent" menu item and choosing a movie file, or by dropping a movie file onto the application's icon. In either case, we know that the movie is associated with an existing movie file. But once we allow the user to create a new movie, we have to make sure not to rely on **NSDocument**'s **fileName** method returning a non-nil value. We'll revise the code in our **initializeMovieWindowFromFile:** method, as shown in **Listing 4**.

Listing 4: Creating a new movie

```

initializeMovieWindowFromFile
if ([self fileName]) {
    // open the movie file with read/write permission and load the movie from it
    err = FSPATHMakeRef([self fileName]
        fileSystemRepresentation], &fileRef, NULL);
    if (err == noErr)

```

Valentina

Object-Relational SQL Database

The fastest database engine for MacOS/Windows

It operates 100's and sometimes
a 1000 times faster than other systems

www.paradigmasoft.com

Hosted by macserve.net
Download full featured evaluation version

Hundreds of innovative new products will be introduced at Macworld!

What Will You Find?

- *RAM: Lifetime Guarantee*
- *Amazing Hard Drives and Enclosures!*
- *Broadest PowerBook Accessory Line!*
- *USB Storage*
- *iPod Essentials*

- *CD-RWs and DVD-R Drives*
- *Connectors & Adapters*
- *Wireless Products*
- *PCI Cards*
- *Great Software*

- *Networking and Analysis Tools*
- *Home Automation*
- *TechToys*
- *Programming Products*
- *Hilarious T-Shirts*

- *FireWire and USB Connectivity*
- *Video & Audio Products*
- *Joysticks, Mice & Keyboards*
- *Server and NetAdmin products*
- *Utilities*
- *Scripting Solutions*

Don't miss out on the hottest products and show specials. Pick up everything you need at DevDepot's trademark truck in the North Hall.

877-DEPOT-NOW • 805/494-9797 • orders@devdepot.com

Why DevDepot?

- *The Hottest products*
- *Special show prices*
- *World Renowned Customer Service*
- *Knowledgeable Staff*

Find them all at special show-only prices
when you visit booth 3761 in the North Hall

The Official
Macworld
Conference & Expo
STORE

Brought to you by...

**DEV
DEPOT**®

www.devdepot.com


```

err = FSGetCatalogInfo(&fileRef, kFSCatInfoNone, NULL,
    NULL, &fileSpec, NULL);

if (err == noErr)
    err = OpenMovieFile(&fileSpec, &fileRefNum,
        fsRdWrPerm);

if (err == noErr)
    err = NewMovieFromFile(&qtMovie, fileRefNum,
        &fileResNum, NULL, 0, NULL);
} else {
    qtMovie = NewMovie(newMovieActive);
}

```

As you can see, we've conditionalized the existing code so that it is called only if the document already has a file associated with it; otherwise, we simply call **NewMovie** to create a new empty movie.

We also need to make sure to set the window title to the last component of the file name only if the file actually has a name:

```

if ([self fileName])
    [[_movieView window] setTitle:[self
        lastComponentOfFileName]];

```

If we do not explicitly set the window title, **NSDocument** will generate one automatically. **Figure 4** shows a new document window with an empty movie.

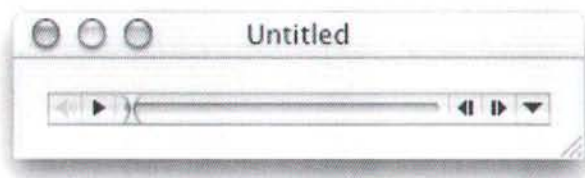


Figure 4: A new, empty document window

We can cut or copy data in other movie windows and then paste it into this new movie. And we can save the new movie into a file, just as we would expect.

Listing 4 is interesting also because it illustrates how to convert a string of type **NSString** into a file system specification (of type **FSSpec**): get a C string representation of the **NSString**, create an **FSRef**, and then call **FSGetCatalogInfo** to get an **FSSpec** record from the **FSRef**. **Listing 5** shows this technique packaged into a nice utility method.

Listing 5: Converting a pathname string into a file system specification

```

(void)NSStringToFSSpec:(NSString *)theFilePath
    fsSpec:(FSSpec *)theFSSpecPtr
{
    FSRef fsRef;
    Boolean isDirectory = false;
    OSStatus err = noErr;

    // create an FSRef for the specified target file
    if (theFilePath) {
        err = FSPathMakeRef([theFilePath
            fileSystemRepresentation], &fsRef, &isDirectory);

        // create an FSSpec record from the FSRef
    }
}

```

```

if (!err && !isDirectory) {
    err = FSGetCatalogInfo(&fsRef, kFSCatInfoNone, NULL,
        NULL, theFSSpecPtr, NULL);
}
}

```

This is useful because Cocoa likes to work with full pathnames, while the QuickTime APIs tend to prefer **FSSpec** records. Note, however, that the **NSStringToFSSpec** function works only with pathnames that describe existing files; if we pass in a path to a nonexistent file, **FSPathMakeRef** will return **fnfErr** (file not found) and fail to create an **FSRef**.

If we want to create an **FSSpec** record for a file that does not exist, we need to be a bit more creative. **Listing 6** defines the **writeToFile:** method, which might be called (for instance) when the user selects the "Save As" menu item. As you can see, if **FSPathMakeRef** returns **fnfErr**, we call the standard UNIX file system functions **open**, **write**, and **close** to create a new file. Then we call **FSPathMakeRef** and **FSGetCatalogInfo** (as above) to get an **FSSpec** record, which we pass to **FlattenMovie**.

Listing 6: Writing a movie into a file

```

(BOOL)writeToFile:(NSString *)path
    ofType:(NSString *)type
{
    FSRef fsRef;
    FSSpec fsSpec;
    NSString *newPath;
    OSStatus err = noErr;

    newPath = [NSString stringWithFormat:@"%s-", path];

    // create an FSRef for the specified target file
    err = FSPathMakeRef([newPath fileSystemRepresentation],
        &fsRef, NULL);
    if (err == fnfErr) {
        // if the file does not yet exist, then let's create the file
        int fd;

        fd = open([newPath fileSystemRepresentation],
            O_CREAT | O_RDWR, 0600);
        if (fd < 0)
            return NO;

        write(fd, " ", 1);
        close(fd);

        err = FSPathMakeRef([newPath fileSystemRepresentation],
            &fsRef, NULL);
    }

    if (err == noErr) {
        // create an FSSpec record from the FSRef
        err = FSGetCatalogInfo(&fsRef, kFSCatInfoNone, NULL,
            NULL, &fsSpec, NULL);
        if (err == noErr) {
            short resId;

            // flatten the movie data into the specified target file
            FlattenMovie([[_movieView movie] QTMovie],
                flattenAddMovieToDataFork |
                flattenForceMovieResourceBeforeMovieData,
                &fsSpec,
                'TVOD',
                smSystemScript,
                createMovieFileDeleteCurFile |
                createMovieFileDontCreateResFile,
                &resId,
                nil);
        }
    }
}

```



```

    CloseMovieFile(resId);
}

rename([newPath fileSystemRepresentation],
       [path fileSystemRepresentation]);
return YES;
}

// clean up
unlink([newPath fileSystemRepresentation]);
return NO;
}

```

WINDOW RESIZING

Let's now take a look at a couple of issues related to resizing our movie windows. Typically our document windows will be resized directly by the user, by dragging the resize box in the lower right corner of the document window. In Interface Builder, we've set the Size attributes of the movie view as shown in **Figure 5**.

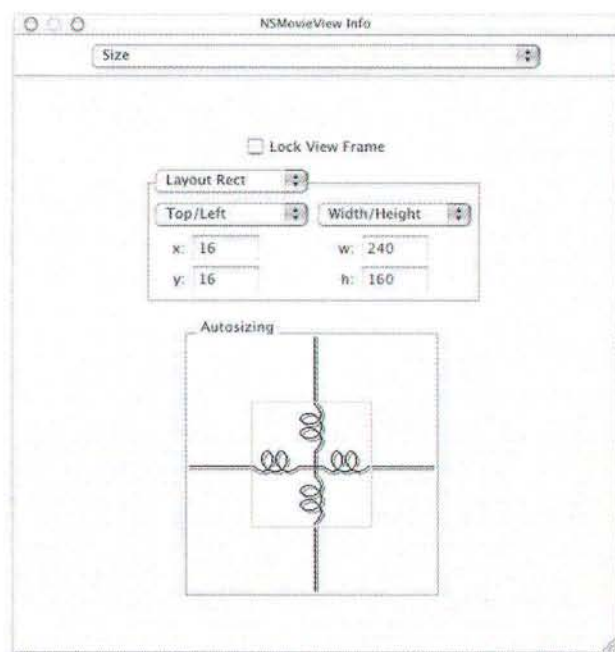


Figure 5: The size attributes of the movie view

The autosizing springs within the movie view and the rigid connections between the movie view and its superview (the document window) indicate that the movie view should grow or shrink to maintain a constant distance from all its edges to the edges of the document window. That is, when the user resizes the document window, the movie view will automatically be resized to maintain its border.

Occasionally, however, we need to adjust the size of the document window based on the desired size of the movie view. We did this previously when setting the size of the window for a newly-opened QuickTime movie. We'll also want to adjust the size of the document window when we receive the

mcActionControllerSizeChanged movie controller action. So the first thing I want to do is rework our existing code a bit, to factor out the code that sets the movie size. **Listing 7** shows our new method **windowContentSizeForMovie**:

Listing 7: Getting a window size from a movie size

```

windowContentSizeForMovie
- (NSSize)windowContentSizeForMovie:(Movie)qtMovie
{
    NSSize size;
    Rect rect;

    GetMovieBox(qtMovie, &rect);
    size.width = (float)(rect.right - rect.left);
    size.height = (float)(rect.bottom - rect.top);

    // enforce a minimum width (important for sound-only movies)
    if (size.width == 0)
        size.width = (float)240;

    size.width += 2 * kMoviePaneOffset;
    size.height += 2 * kMoviePaneOffset;

    if ([_movieView isControllerVisible])
        size.height += kMovieControllerBarHeight;

    return size;
}

```

When the movie window is awakened from the nib file, we set the movie window size like this:

```

[_movieView window] setContentSize:
    [self windowContentSizeForMovie:qtMovie];

```

high quality - competitive rates - 16 years experience - award winning

Full Spectrum Software

Development & Testing

Device Drivers
Porting
Plug-ins

TCP/IP

Carbon / OSX

Cross Platform Development

One Bridge Street
Newton, MA 02458

617-965-0029

www.FullSpectrumSoftware.com

competitive rates - 16 years experience - award winning - high quality

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

And when we receive the `mcActionControllerSizeChanged` movie controller action, we can set the movie window size as shown in **Listing 8**.

Listing 8: Setting a window size from a movie size

```
MyActionFilter
case mcActionControllerSizeChanged:
    [[[doc movieView] window] setContentSize:
        [doc windowContentSizeForMovie:[[[doc movieView]
        movie] QTMovie]]];
    break;
```

Now, when do we receive the `mcActionControllerSizeChanged` action? We'll receive it whenever the user manually resizes our document window, since `NSMovieView` informs the movie controller — probably by calling `MCSetControllerBoundsRect` — whenever the movie view is resized. (In this case, we really don't need to call `setContentSize:`, but it doesn't hurt to do so.) We'll also receive the `mcActionControllerSizeChanged` action when certain wired actions change the movie size. For instance, QuickTime 6 introduced the `kActionMovieSetScale` action, which sets the target movie's scale (or magnification). And earlier versions of QuickTime included the `kActionTrackSetMatrix` wired action, which sets a track's matrix. When the movie controller processes either of these actions, it will eventually inform our application of the size change by sending the `mcActionControllerSizeChanged` action to our filter procedure.

It turns out that the code in **Listing 8** can lead to some drawing glitches when triggered by these wired actions. When a movie receives the `kActionMovieSetScale` action, our window can end up looking like the one in **Figure 6**. As you can see, the controller bar was not correctly erased at its previous position or redrawn fully in the new position.



Figure 6: Drawing problems after a set scale action

I haven't investigated this behavior enough to know whether it's a problem in QuickTime's action-handling code or in our own code. But there is at least one easy workaround: just inform the

BMS

**THE LAW OFFICE OF
BRADLEY M. SNIDERMAN**

Need help safeguarding your software?

If you're developing software, you need your valuable work protected with trademark and copyright registration, as well as Non Disclosure Agreements.

Then, when you are ready to sell it, you can protect yourself further with a licensing agreement.

I am an attorney practicing in Intellectual Property, Business Formations, Corporate, Commercial and Contract law.

Please give me a call or an e-mail. Reasonable fees.

23679 Calabasas Rd. #558 • Calabasas, CA 91302

PHONE 818-222-0365 FAX 818-591-1038 EMAIL brad@sniderman.com

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."

– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."

– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."

– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"

– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."

– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."

– Joe Zobkiw, author of A Fragment of Your Imagination

"Resorcerer will pay for itself many times over in saved time and effort."

– MacUser review

"The template that disassembles PICT's is awesome!"

– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensable in its own creation!"

– Doug McKenna, author of Resorcerer



Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: \$256 (decimal)

Website price: \$128 - \$256

(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMAESTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New
in
2.0:

- Very fast, HFS browser for viewing file tree of all volumes
- Extensibility for new Resorcerer Apprentices (CFM plug-ins)
- New AppleScript Dictionary ('aete') Apprentice Editor
- MacOS 8 Appearance Manager-savvy Control Editor
- PowerPlant text traits and menu command support
- Complete AIFF sound file disassembly template
- Big-, little-, and even mixed-endian template parsing
- Auto-backup during file saves; folder attribute editing
- Ships with PowerPC native, fat, and 68K versions

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

movie controller that the movie has changed, by calling **MCMovieChanged**. Listing 9 shows our updated code for handling the **mcActionControllerSizeChanged** action.

Listing 9: Setting a window size from a movie size (revised)

```
MyActionFilter
case mcActionControllerSizeChanged:
    [[[doc movieView] window] setContentSize:
        [doc windowContentSizeForMovie:[[[doc movieView]
            movie] QTMovie]]];
    MCMovieChanged([[[doc movieView] movieController],
        [[[doc movieView] movie] QTMovie]]);
    break;
```

By the way, this is not a Cocoa-specific problem. It will also affect our Carbon-based QTShell application. Accordingly, we should add a call to **MCMovieChanged** in QTShell's movie controller action filter procedure as well, as shown in Listing 10.

Listing 10: Setting a window size from a movie size (QTShell)

```
QTApp_MCActionFilterProc
case mcActionControllerSizeChanged:
    QTFrame_SizeWindowToMovie(myWindowObject);
    if (theMC && (**myWindowObject).fMovie)
        MCMovieChanged(theMC, (**myWindowObject).fMovie);
    isHandled = true;
    break;
```

There is at least one other occasion when the movie controller might send us the **mcActionControllerSizeChanged** action, namely when a streamed movie changes its size dynamically during playback. We don't need any additional code to handle this, but we do need to tell the Movie Toolbox that we want to be informed of any changes in the size of streamed movies. We do this by setting a movie playback hint when we open the movie, as follows:

```
SetMoviePlayHints(qtMovie, hintsAllowDynamicResize,
    hintsAllowDynamicResize);
```

We are informed of movie size changes triggered by wired actions even if we don't set this play hint, but not those triggered by a dynamic size change of a streamed movie.

CURSOR ADJUSTMENT

As you know, some media types change the cursor image as it moves over various regions in a track. Flash tracks and wired sprite tracks often do this, and QuickTime VR does this as a matter of course. (And with a vengeance: QuickTime VR defines nearly 80 different cursors that can be displayed as the user performs operations within a panoramic movie, and over 100 within an object movie.) Figure 7 shows a QuickTime VR movie with the mouse lying on top of a link hot spot (that is, a hot spot that, when clicked, moves the user to a new node).

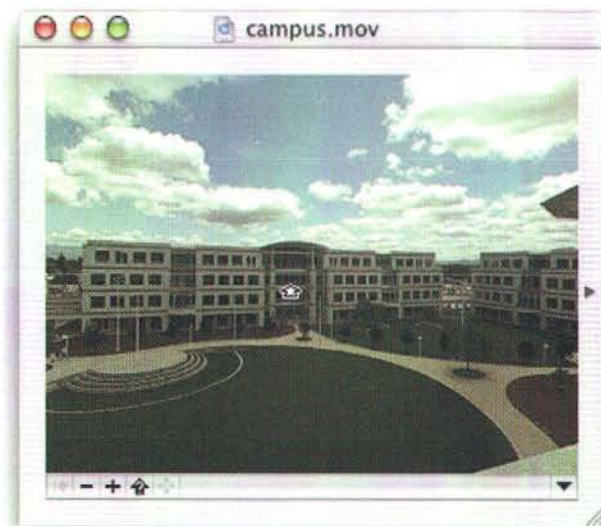


Figure 7: A QuickTime VR node with the mouse over a link hot spot

The trouble is that some media types don't change the cursor back to the default arrow cursor when it moves outside of the movie box. The result is that our application ends up displaying an incorrect cursor, as shown in Figure 8. The mouse is just to the right of the movie box (toward the top of the movie). Note that its image is still one supplied by QuickTime VR, not the standard arrow cursor. We should fix that.



Figure 8: A QuickTime VR cursor displayed outside the movie box

Adjusting the Cursor in Carbon

Before we see how to adjust the cursor in our Cocoa application, let's digress briefly to consider how we already do

The power of UNIX brought to the Mac

mac:ODBC

Open database connectivity
for the Macintosh

Advanced SQL Editor

Powerful multi-platform
and multi-DB SQL script editor

SQL Project

Multi-platform SQL
development environment

Team Diagram

Dynamic diagramming and charting
tool with CVS and database hooks

Data Architect

Multi-platform suite of database
modelling and design tools: includes
all the above software components

More Mac OS X products from
theKompany.com coming soon

Take advantage of the power of UNIX with theKompany.com's suite of database software. Now also for Mac OS X.

TheKompany.com, a UNIX software and services company, now offers its powerful UNIX database connectivity tools for the Mac OS X computing platform.

With theKompany.com's suite of database tools, you can do just about anything with your SQL databases: connect to SQL databases with mac:ODBC, create and edit SQL scripts with the Advanced SQL Editor, create diagrams from CVS trees and databases with Team Diagram, develop powerful SQL-based projects with SQL Project and more.

Each of these tools is available separately, or as part of Data Architect – theKompany.com's premiere database modelling tool. All are offered with a unique open source license, so you can tweak the program code to your liking.

Prices for these great products start at just **\$9.95** for individual packages and top out at **\$49.95** for the complete Data Architect suite.

For more information about pricing and availability, to try a demo version, or to place your order online, visit us at www.thekompany.com.

theKompany.com



this in our Carbon application QTShell. **Listing 11** shows the definition of the **QTApp_Idle** function, which we call whenever we receive an idle event. As you can see, we simply check whether the current mouse position (which we get by calling **GetMouse**) is outside of the front movie window or front movie window's visible region; if the mouse is indeed outside of these regions, we call **MacSetCursor** to reset the cursor to the arrow cursor.

Listing 11: Adjusting the cursor in an idle procedure

```
void QTApp_Idle (WindowReference theWindow)
{
    WindowObject      myWindowObject = NULL;
    GrafPtr            mySavedPort;

    GetPort(&mySavedPort);
    MacSetPort(QTFrame_GetPortFromWindowReference(theWindow));

    myWindowObject = QTFrame_GetWindowObjectFromWindow
        (theWindow);
    if (myWindowObject != NULL) {
        MovieController myMC = NULL;

        myMC = (**myWindowObject).fController;
        if (myMC != NULL) {
            #if TARGET_OS_MAC
                // restore the cursor to the arrow
                // if it's outside the front movie window or outside the window's visible region
                if (theWindow == QTFrame_GetFrontMovieWindow()) {
                    Rect      myRect;
                    Point      myPoint;
                    RgnHandle myVisRegion;
                    Cursor      myArrow;

                    GetMouse(&myPoint);
                    myVisRegion = NewRgn();
                    GetPortVisibleRegion
                        (QTFrame_GetPortFromWindowReference(theWindow),
                         myVisRegion);
                    GetWindowPortBounds(theWindow, &myRect);
                    if (!MacPtInRect(myPoint, &myRect) ||
                        !PtInRgn(myPoint, myVisRegion))
                        MacSetCursor(GetQDGlobalsArrow(&myArrow));

                    DisposeRgn(myVisRegion);
                }
            #endif
        }
    }
}
```

```

    }
#endif
}

MacSetPort(mySavedPort);
}
```

Frankly, this is old-style Mac programming. It works well enough, but it's likely to eat up CPU cycles unnecessarily since it's still stuck in that old polling mindset. Let's upgrade QTShell by reimplementing cursor adjusting using Carbon events. (For a general discussion of Carbon events and QuickTime, see "Event Horizon" in *MacTech*, May 2002.)

My first attempt to use Carbon events here was to have the window event handler register for events of class **kEventClassMouse** and type **kEventMouseMoved**. After all, we need to check to see if the cursor needs to change only if it's been moved. The trouble is that a movie in QTShell completely fills the content region of a movie window (except for the space occupied by the controller bar at the bottom of a movie window). Once the cursor moves out of the movie to the left or right, it's no longer over the movie window and hence subsequent mouse movements will not trigger **kEventMouseMoved** events for that movie window.

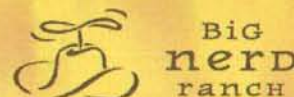
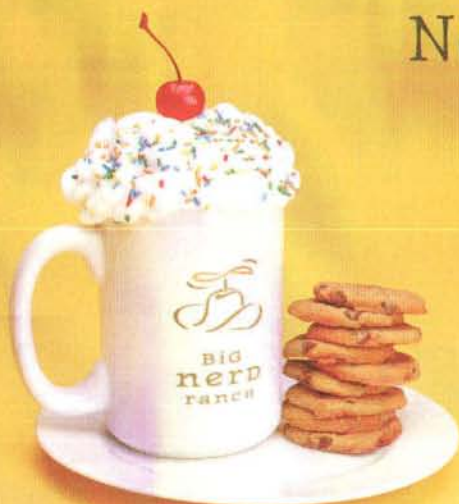
So I ended by having the application event handler register for **kEventMouseMoved** events, as shown in **Listing 12**. When it receives a mouse-moved event, it retrieves the position of the mouse and then checks to see if the mouse is currently outside the frontmost movie window. If it is, the cursor is reset to the default arrow cursor.

Listing 12: Adjusting the cursor in a Carbon event handler

```
QTFrame_CarbonEventHandler
case kEventClassMouse:
    switch (myKind) {
        case kEventMouseMoved:
```

Now serving Cocoa[®] just the way you want it.

Training for Mac OS X doesn't have to be the same old flavor. Reserve your seat in a class at our scenic lodge location, or have experts come to you for **Extreme Mentoring**. Two weeks of on-site instruction and collaboration, customized to the requirements of your project. Book now for 2003. See why we're different.



Intensive Classes for Programmers
www.bignerdranch.com

[illegible]

The image shows the box for 'FastTrack 8 Schedule' software. The box is blue and features a large, stylized graphic of a red and blue key. The text 'FastTrack 8' is prominently displayed in a large, white, sans-serif font, with 'Schedule' written below it in a smaller, white, sans-serif font. There are also some smaller logos and text on the box, including a 'New' badge in the top left corner.



Long Distance

3.9¢ Per Minute!

Straight 6 second billing increments

Excellent rates on intrastate, intralata/toll calls and international calling with no term contract.

Toll Free (800/888/877/866) service, same low per minute rate for incoming calls.

10 cents per minute calling card.

Detailed billing directly from Capsule Communications, a Covista Company.

**Quality electronic and telephone customr support.
No monthly billing fee if you sign up for AUTOPAY billing option or if your bill is over \$20.00 each month.**

(NOTE: \$1.00 billing fee is charged when your bill is under \$20.00 for all non-Autopay customers.)



www.lowcostdialing.com

```
myErr = GetEventParameter(theEvent,
    kEventParamMouseLocation, typeQDPoint, NULL,
    sizeof(Point), NULL, &myPoint);
if (myErr == noErr) {
    WindowRef myWindow = NULL;
    Rect myRect;
    Cursor myArrow;

    GlobalToLocal(&myPoint);

    // get the front movie window
    myWindow = QTFrame_GetFrontMovieWindow();
    if (myWindow != NULL) {
        GetWindowPortBounds(myWindow, &myRect);
        if (!PtInRect(myPoint, &myRect))
            MacSetCursor(GetQDGlobalsArrow(&myArrow));
    }

    break;
}
break;
```

This strategy appears to work quite nicely, and it avoids the polling behavior of our original code. Strictly speaking, we need to adjust the cursor only when a movie contains interactive tracks (that is, Flash, wired sprite, wired text, or QuickTime VR), since they are the only kinds of tracks that are likely to change the cursor from the default arrow cursor. I doubt, however, that there is much to be gained by checking for interactive track types here. It's probably better just to reset the cursor to the arrow for every kind of QuickTime movie, as we do here.

Adjusting the Cursor in Cocoa

Let's return to the more pressing concern of adjusting the cursor in our Cocoa application. There are two principal ways we can return the cursor to its default arrow shape when it moves outside of the movie rectangle. First, we can add a *tracking rectangle* to the **NSMovieView** view that holds our movie, by executing the **addTrackingRect:** method. For instance, inside of **windowControllerDidLoadNib:**, we can execute this line of code:

```
[_movieView addTrackingRect:[_movieView bounds] owner:self
    userData:nil assumeInside:NO];
```

The **addTrackingRect:** method causes the specified owner to receive **mouseEntered:** and **mouseExited:** messages whenever the mouse is moved into and out of the specified rectangle inside the target view. The message recipient is often the same view whose rectangle will be tracked, but it need not be. In the present case, indeed, we are setting the document instance as the owner, so that these messages are sent to it and not to the movie view. This allows us to define **mouseEntered:** and **mouseExited:** within our custom document class, so that we do not need to subclass **NSMovieView**.

Actually, since we are only interested in knowing when the mouse leaves the movie view, we shall implement only the **mouseExited:** method (Listing 13).

Listing 13: Handling mouse-exited messages

```
- (void) mouseExited:(NSEvent*)event mouseExited
{
    // set cursor to the default cursor
```



```
[[NSCursor arrowCursor] set];
```

Here we send the factory method `arrowCursor` to the `NSCursor` class and then set the resulting cursor.

It turns out that `windowControllerDidLoadNib:` is not really the best place to call `addTrackingRect:`. The main reason for this is that the tracking rectangle is not automatically resized when the target view is resized. Rather, we need to explicitly remove any current tracking rectangle and then add a new one each time the movie view changes size. So let's make the call to `addTrackingRect:` inside of our movie controller action filter procedure, when we handle the `mcActionControllerSizeChanged` action. **Listing 14** shows our revised code for handling this action.

Listing 14: Resetting the tracking rectangle

```
MyActionFilter
case mcActionControllerSizeChanged:
    [[doc movieView] window] setContentSize:
        [doc windowContentSizeForMovie:
            [[[doc movieView] movie] QTMovie]];
    [[doc movieView] removeTrackingRect:
        [doc trackingRectTag]];
    [doc setTrackingRectTag:[doc movieView]
        addTrackingRect:[doc movieView] bounds] owner:doc
        userData:nil assumeInside:NO];
    MCMovieChanged([doc movieView] movieController),
        [[[doc movieView] movie] QTMovie]);
    break;
```

First, we send the `removeTrackingRect:` message to the movie view to remove any existing tracking rectangle. Notice that `removeTrackingRect:` takes a parameter, which specifies a *tracking rectangle tag* (of the scalar type `NSTrackingRectTag`). This tag is returned to us by `addTrackingRect:`, and we are storing it in the new instance variable `_trackingRectTag` in our document class. Since we are calling `removeTrackingRect:` and `addTrackingRect:` within our movie controller action filter procedure, the document class needs to provide accessor methods for us to get and set the tag. **Listings 15** and **16** show our definitions of these two methods. Nothing earthshaking here.

Listing 15: Getting the tracking rectangle tag

```
trackingRectTag
- (NSTrackingRectTag)trackingRectTag
{
    return _trackingRectTag;
}
```

Listing 16: Setting the tracking rectangle tag

```
setTrackingRectTag
- (void)setTrackingRectTag:(NSTrackingRectTag)rectTag
{
    _trackingRectTag = rectTag;
}
```

The second way of adjusting our cursor is to define a *cursor rectangle* for the movie view. A cursor rectangle is a special kind of tracking rectangle. What's special about it is that `NSView` knows that the rectangle is likely to change if the associated view is resized or moved. Accordingly, `NSView` calls the view's `resetCursorRects` method whenever that

PRIMEBASE 4.0

New Launcher for OS X

Installation has never been easier on Mac OS X with the PrimeBase Launcher.

Download the Launcher **now** and get started with PrimeBase

www.primebase.com

**developer keys
available
free of charge**

**PRIMEBASE DATABASE SERVER AND
PRIMEBASE APPLICATION SERVER
AVAILABLE ON THE MOST POPULAR PLATFORMS**

- Completely cross-platform
- Full-text searching and indexing
- Mac OS classic & X
- Mac OS X Server
- Linux
- Solaris
- IBM AIX
- all Windows platforms

Develop on a Mac and deploy without any additional effort on any platform you want.

 **PRIMEBASE**

SNAP Innovation GmbH
Altonaer Poststraße 9a
D-22767 Hamburg / Germany
www.primebase.com
e-mail: info@primebase.com
Fon: ++49 (40) 389 044-0
Fax: ++49 (40) 389 044-44

happens, to give the view a chance to resize or move the cursor rectangle. **Listing 17** shows how we might define the `resetCursorRects` method.

Listing 17: Resetting the cursor rectangle

```
resetCursorRects
- (void)resetCursorRects
{
    NSCursor *newCursor;

    [super resetCursorRects];

    newCursor = [NSCursor arrowCursor];
    [self addCursorRect:[self bounds] cursor:newCursor];
    [newCursor setOnMouseExited:YES];
}
```

Here we create a new arrow cursor and call `addCursorRect:` to attach it to a new cursor rectangle that is as large as the target view (that is, the movie view). Then we call `setOnMouseExited:` to configure the cursor to set itself as the current cursor when the mouse moves outside of the cursor rectangle.

As you can see, there is much less code required when using cursor rectangles than when using tracking rectangles. We don't need to keep track of the tracking rectangle tag, and we don't need any accessor functions to pass that tag to the movie controller action filter procedure. The only downside to using

cursor rectangles is that we must subclass `NSMovieView` (so that we have a class to define the `resetCursorRects` method). As we saw above, we did not need to subclass `NSMovieView` when using tracking rectangles.

CONCLUSION

In this article, we revisited the basic Cocoa movie playing and editing application that we developed in the previous article, with an eye to tying up a few loose ends. We've now got all the items in the File and Edit menus working as expected, and we've cleaned up a few cosmetic glitches in our original version of MooVeez. With these improvements, MooVeez now matches quite closely the capabilities of QTShell, our benchmark Carbon movie playback and editing application. Moreover, the Cocoa framework upon which MooVeez is built provides a number of additional features not found in QTShell, including the "Open Recent" menu item in the File menu and the Window menu for managing all open document windows.

CREDITS

Listing 6 is based on some code by Vince DeMarco.

ListSTAR®

www.liststar.com

The most flexible email processing system available. Easily create mailing lists or email-on-demand services. Use built in rules and/or AppleScript/AppleEvents to handle any email task, no matter how simple or complex. Demo available.

MacRADIUS™

www.macradius.com

The easiest to use RADIUS server available. The groups feature allows you to make changes for a large number of users in one easy step. Enabling or disabling access for a user or a group of users is a one click operation, without having to stop the server. Support for AppleScript/AppleEvents makes it easy to control MacRADIUS. Demo available.

SimpleText Filter

Plug-in for EIMS*
www.mcfsoftware.com/stf/

Easy to use header and content filtering. Scan incoming messages for sequences of text, digits, etc. Base64 messages are decoded so you can check for content despite attempts to hide the text. Demo available.

Auto Reply

Plug-in for EIMS*
www.mcfsoftware.com/ar/

This plug-in allows you to easily set up auto-reply messages for users. Addresses that have been sent auto-reply messages are tracked, preventing auto-reply message loops. Demo available.

Address List Sorter

www.mcfsoftware.com/als/

Fast and powerful utility for sorting and cleaning email lists. Sort email address lists in alphabetical or domain order, remove duplicates and improperly formed addresses. Demo available.

*EIMS—Eudora Internet Mail Server (www.eudora.co.nz)



...*simply*
dependably
engineered

www.mcfsoftware.com

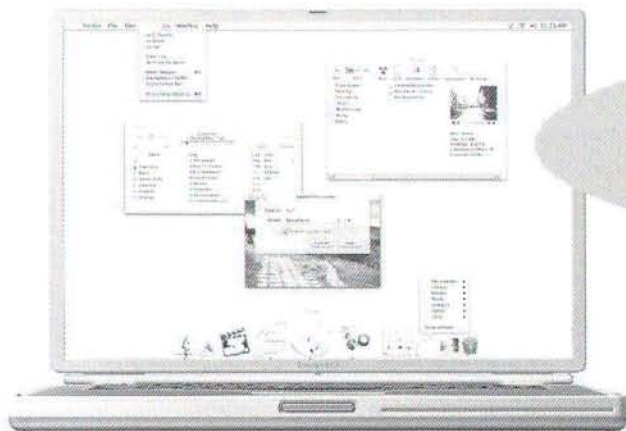
Special Small Dogs

Here are just a few of the photos our customers have sent us of their special dog friends!



Check out all the other special Small Dog photos and send us one of your own at www.smalldog.com!

Small Dog's Special



PowerBook G4/667 Titanium
\$2499

- 512MB RAM
- 30gb Hard Drive
- Combo Drive
- Airport card and base station



Purchase from an Apple Specialist!

Small Dog Electronics is proud to have earned Apple's prestigious designation of Apple Specialist. Small Dog Electronics has exceeded all of Apple's stringent requirements for its Specialists and has taken it one step further.

Small Dog sells only the most powerful personal computers in the world—every single one of them an Apple Macintosh! In addition to the rigorous Apple training program, each Small Dog employee has chosen a specific area of expertise to concentrate upon. You can be assured that whomever you talk to at Small Dog, from our sales engineers to our shipping department, you are talking to a trained, enthusiastic, Apple Macintosh Specialist!

Small Dog Electronics is also an Authorized Apple Service Provider Plus. We have certified Apple Technicians that utilize genuine Apple parts for all repairs and provide technical support for our customers based upon their experience in upgrading and supporting thousands of Apple Macintosh computers for our customers!

Talk To an Apple Professional!

Did you know that when you call Small Dog Electronics, you are most likely talking to an Apple Product Professional? Each year, we participate in Apple's on-line courses and seminar training programs to learn as much as we can about the products that we sell and service!

Exclusive Top Dog Membership Treats

For each dollar purchased on-line at the Small Dog web site, you will receive a doggie treat. You can redeem your accumulated treats for Small Dog or Apple apparel and other products. You are automatically enrolled and begin accumulating treats in the Top Dog Club with your first purchase. Remember the more you buy, the more treats for you!



**Small Dog
Electronics**

1673 Main Street
Waitfield, VT 05673 USA
Phone: 802-496-7171
Online: smalldog.com



By Rich Morin

CamelBones

Creating GUI-based apps with Perl

Sherm Pendley bills CamelBones as "An Objective-C/Perl bridge framework". The web page goes on to say "CamelBones is a framework that allows many types of Cocoa programs to be written entirely in Perl. It also provides a high-level object-oriented wrapper around an embedded Perl interpreter, so that Cocoa programs written in Objective-C can easily make use of code and libraries written in Perl."

Jumping from the general to the particular, CamelBones allows a Perl scripter to create GUI-based apps, with only a modicum of pain (read, Objective-C :-). Because the scripter can use Interface Builder, the graphic layout is painless, quick, and likely to yield attractive results.

LIMITATIONS

Sherm says that CamelBone's biggest current limitation is inheritance. It's not yet possible to create a Perl sub-class that inherits from an Objective-C sub-class. As a result, CamelBones can't be used for:

- Document-based applications, which require a subclass of `NSDocument`.
- Custom controls and/or cells, which require a subclass of one of the many `NSControl` or `NSCell` descendants, respectively.

This limitation will be addressed in the next version (0.3), which Sherm expects to have ready by the time this article is in print. Meanwhile, most of the tasks which could be performed by a sub-class can be handled by a `Delegation` or `Notification` callback.

CamelBones has some other problems, at least from my perspective. Because the app needs to be "built" (with Project Builder) rather than simply run, the build time becomes a noticeable part of the edit/test cycle. I think that's just the price we pay for using compilers and linkers (-:-).

Also, most of the existing Cocoa documentation assumes that you're using Objective-C. This means that you have to

have a reading knowledge of Objective-C, as well as the ability to turn method prototypes and example code into their Perl equivalents. I find this livable, however, and help is on the way for both problems

GETTING STARTED

Assuming that you already have OSX and Apple's Developer Tools installed, adding CamelBones is quite simple. Go to <http://camelbones.sf.net>, download the distribution, and install it! At this writing, the documentation is still a bit sketchy, but the author plans to improve it Real Soon Now, so it may be in significantly better shape by the time you see it.

The package uses a standard installer, so installation is quite easy. I'd suggest that you choose the "Custom" install, so that you can get the source code (why not?). I haven't looked at the code, but I assume that it's a combination of Objective-C and Perl. Nice to have around, if you get frustrated by some peculiarity or simply become curious about how it all works.

If you are thinking about using CamelBones in a proprietary offering, you may want to look over the license a bit. CamelBones is released under the GNU Project's "Lesser General Public License" (LGPL). Briefly, the LGPL allows your app to use CamelBones without any requirement that the author provide the source code for the application (though with Perl, source distribution is the default case). On the other hand, any changes you make to CamelBones itself must be made available to anyone who gets the "binaries".

Once you have CamelBones installed, I strongly suggest that you skim the documentation and walk through the `HowTo` examples. As a newcomer to Interface Builder and Project Builder, this taught me a bit about the capabilities and operation of each. Even if you are expert at IB and PB, the examples will give you an idea of what kind of Perl code CamelBones expects.

In brief, however, Perl code for use with CamelBones looks pretty much like Perl code to work with any object-oriented API. Stuff like:

```
sub sayHello {
    my ($self, $sender) = @_;
```

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.

© 2002 Microsoft Corporation. All rights reserved. Microsoft, Entourage, PowerPoint, Word, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple, Macintosh, and Mac OS are registered trademarks of Apple Computer, Inc.

Microsoft



Macs and PCs have never been so compatible.

Microsoft® Office v. X lets Mac users effortlessly open, edit, and save any Office file, to make working with PCs a breeze. Complete with easy-to-use, exclusive Mac tools that simplify complex tasks. And it's built for Mac OS X, so it's the most reliable, easygoing Office yet. Go www.officeformac.com to download a free 30-day trial of Office v. X today.



Microsoft
Office:mac
v. X


```
$self->[UILabel] ->setStringValue("Hello");
NSLog("Hello, world!");
}
```

One peculiarity, discussed in the CamelBones web pages, is that CamelBones doesn't (yet) provide "toll-free" bridging of Perl hashes and lists to Cocoa's collection classes. Consequently, the Perl code has to deal with the Cocoa collections explicitly. Instead of writing:

```
my %hash;
my $key = 'abc';
$hash{$key} = 'def';
printf("key=%s, value=%s\n",
    $key, $hash{$key});
```

you have to write something like:

```
my $hash = NSMutableDictionary ->alloc ->init;
my $key = 'abc';
$hash->setObject_forKey($key, 'def');
printf("key=%s, value=%s\n",
    $key, $hash->objectForKey($key));
```

Also, because Objective-C doesn't provide automatic (Perl-style) garbage collection, you may need to explicitly reserve and release any Objective-C objects which you create. This looks pretty tedious, to my Perl-accustomed eyes, but I'm consoled by the facts that (a) I only have to use Cocoa collections to access Cocoa-specific items and (b) relief is said to be on the way.

PACKAGING ISSUES

Ease of distribution and installation has been a CamelBones priority from the start. All that the end user needs is a copy of CamelBones.framework. By default, applications look for this in /Library/Frameworks, so the easiest thing to do is to create an installer package that will make sure that the framework can be found there.

But, if a developer wants a drag-and-drop install and is willing to rebuild the framework with the appropriate Project Builder options, the framework can be embedded in the application bundle itself. CPAN (Comprehensive Perl Archive Network) modules can also be included in the application bundle, eliminating another common source of pain for end users of Perl programs.

VERSION CREEP

The CamelBones framework is linked against the Perl interpreter (Version 5.6.0) that is shipped with OSX. Sherm has gotten reports that Perl 5.6.1 works fine, as long as it's compiled and installed identically to the original 5.6.0, but 5.8.0 definitely doesn't. So, if you have added Perl 5.8.0 to your system, you may have to rebuild any CamelBones apps you receive.

As time goes on, Apple is quite likely to release Perl 5.8.0 (or whatever) as an update to OSX. Unless a workaround is found, this will cause all 5.6.0-based

CamelBones apps to break. Fortunately, some Very Bright People are looking into the matter, so a fix is likely.

RESOURCES

Perl wizard Dan Sugalski is currently writing "Programming Cocoa Applications with Perl" for O'Reilly, but a publication date has not yet been established. The book will cover CamelBones in depth, however, so watch for it! In the meanwhile, Dan promises to put up some example code on the CamelBones web site.

Notwithstanding the fact that they assume Objective-C usage, most books on Cocoa, IB, and PB are relevant to CamelBones. Here are some you might want to look over:

"Building Cocoa Applications: A Step-by-Step Guide"

Garfinkel and Mahoney
O'Reilly and Associates, 2002
ISBN 0-596-00235-1

"Cocoa Cookbook for Mac OS X"

Bill Cheeseman
Peachpit, 2002
ISBN 0-201-87801-1

"Cocoa Programming for Mac OS X"

Aaron Hillegass
Addison-Wesley, 2001
ISBN 0-201-72683-1

"Learning Cocoa with Objective-C", second edition


James Duncan Davidson
O'Reilly and Associates, 2002
ISBN 0-596-00301-3

There is also a 200+ page rundown on Objective-C, right on your Mac OS X system (/Developer/Documentation/Cocoa/ObjectiveC/ObjC.pdf). You should also consider joining the MacOSX-Perl email list, which covers CamelBones and other Perl-ish topics on Mac OS X. Send email to macosx-subscribe@perl.org to get started.

Although Apple provides documentation on the AppKit and Foundation frameworks, it can be tedious to navigate. So, pick up a copy of Hoshi Takanori's Cocoa Browser app (<http://homepage2.nifty.com/hoshi-takanori/cocoa-browser>). This will let you browse the Objective-C method descriptions in a speedy, hierarchically-based manner.

As noted above, you will still have to convert the method synopses into Perl format, but that's life. Actually, I'm actually working on a conversion app, but you'll have to wait until next month to read about it. Until then, happy hacking...

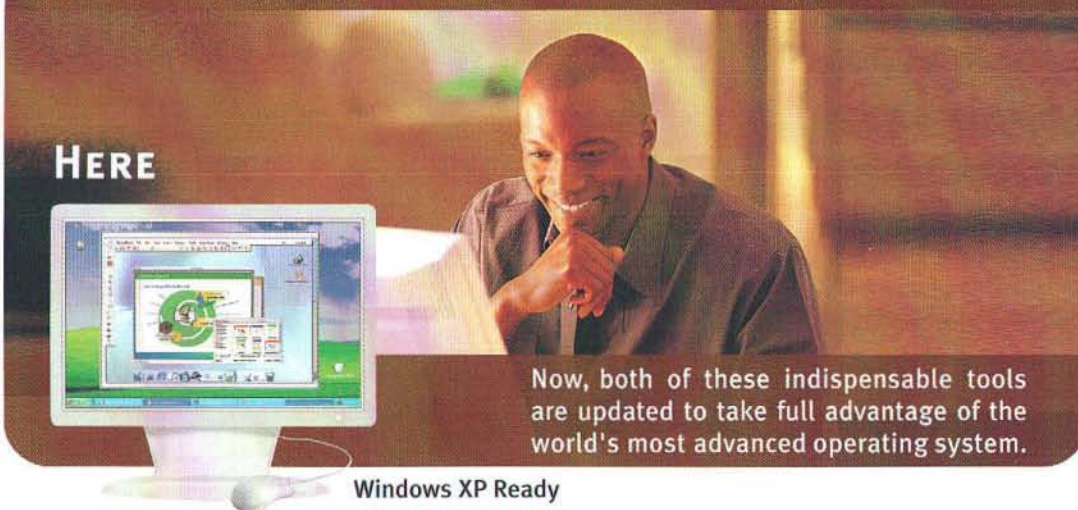
Stay In Control Wherever You Go.



THERE

For more than a decade, Timbuktu Pro and netOctopus have been the leading remote control, file transfer and systems administration applications for the Mac OS.

Mac OS X Ready



HERE

Now, both of these indispensable tools are updated to take full advantage of the world's most advanced operating system.

Windows XP Ready

Timbuktu Pro

Whether you're at home or at work, Timbuktu Pro allows you to operate distant computers as if you were sitting in front of them, transfer files or folders quickly and easily, and communicate by instant message, text chat, or voice intercom.

<http://www.timbuktopro.com>

netOctopus

Intuitive and powerful, netOctopus can manage a network of ten or 10,000 computers. Inventory computers, software and devices on your network; distribute software; configure remote computers; and create custom reports on the fly.

<http://www.netoctopus.com>

Learn more, try it, or buy it online. Call us at **1-800-485-5741**.



timbuktu® • netOctopus®

netopia.

by Danny Swarzman

TicTacPalm 2

Saving Data in a Palm OS Application

INTRODUCTION

In a previous article ("TicTacPalm: Getting Started with Palm OS" in *MacTech* April, 2002), I presented a basic Palm OS application for a person to play Tic-Tac-Toe against the handheld computer. That article presented the structure of an application and the elements of the user interface. Here, we'll go one step further, adding the ability to save and restore documents.

On the Palm OS, each application can have one or more databases associated with it. Our sample application has only one database. Each record in the database is a *game record*. The user can review and modify saved games. This article shows how to save and restore game records. A future article will show how the game data can be transferred to a desktop computer.

THE APPLICATION

TicTacPalm has three forms, a main game board form, a game list form, and game info form. The main form of the application is used both to enter new moves into a game and to review a game record. We use tape-recorder style buttons to review a game, moving forward or backwards. They are visible or hidden as needed. **Figure 1** shows the board when reviewing a game.

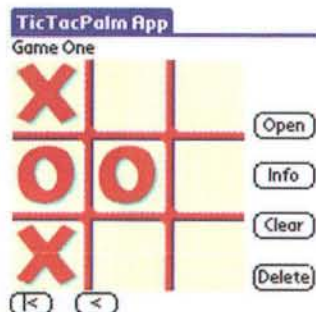


Figure 1: The Game Board Form

The game list form has a scrolling list of names of games. The user selects a game to open or taps the New button to start a fresh game. (See **Figure 2.**) As you can see, there are buttons to open a game and to delete a game.

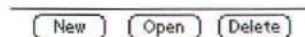


Figure 2: Game List Form

When the user deletes a game, the record doesn't completely disappear from the database. Instead, the record is marked for deletion. The record is eliminated when the next Hot Sync occurs. (We'll discuss this in greater detail in another article — about conduits.)

In the game info form, the user enters the name that is to be associated with the game. The name doesn't need to be unique. The program distinguishes between games according to a record number.

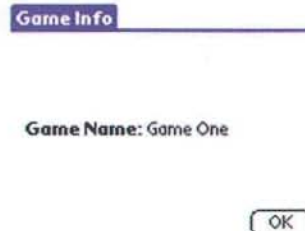


Figure 3: Game Info Form

When the user taps OK in this form, control returns to the game board.

Figure 4 shows how the buttons can be used to navigate among the various forms. Menus could have been used instead of buttons. Menus require more effort to use, but they are needed when the application is more complex.

Danny Swarzman writes programs in JavaScript, Java, C++, and other languages. He also plays Go and grows potatoes. You can contact him with comments and job offers at dannys@stowlake.com, or you can visit his web site at <http://www.stowlake.com>.

Ask yourself this question...

I use my Mac for:

- ☐ Programming as a hobby
- ☐ Exploring the depths of Mac OS X
- ☐ QuickTime Development
- ☐ Application Development
- ☐ Network Administration
- ☐ All Of The Above

...this is the answer:

 **MacTech[®]**

The Journal of Macintosh Technology and Development

Whether you program as a hobby or are a full-time developer, MacTech gives you the info you need from the people that know.

www.mactech.com

PO Box 5200 • Westlake Village, CA • 91359-5200 • orders@mactech.com
Toll-Free: 877-MACTECH • Outside US/Canada: 805-494-9797 • Fax: 805-494-9798

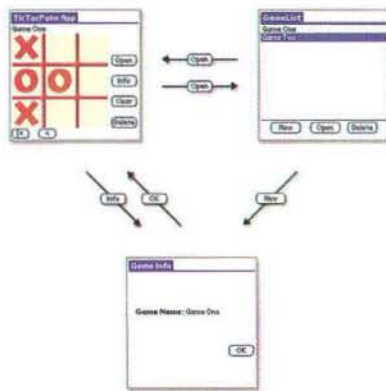


Figure 4: Links Among Forms

DATABASES

Creator and Type

A database on the Palm OS is a set of records associated with a creator and type. A *creator* is the 32-bit code corresponding to the application. An application can have several databases. The *type* is another 32-bit code that an application can use to distinguish among its databases.

Records

A record can be any size up to 64k bytes. Applications in which documents are larger must segment the documents. Palm OS does have a file system, which uses the Data Manager and is not particularly fast. Each record has flags that are maintained by the Data Manager and accessed through Data Manager functions.

- The *delete* flag indicates that the user has deleted a record on the Palm OS device. When Hot Sync is performed, the file will be deleted on the desktop machine and finally be eliminated from the Palm OS device.
- The *dirty* flag indicates that the record has been modified since the last Hot Sync.
- The *busy* flag locks a record for writing.
- The *secret* flag is cleared only when the user password has been entered.

Game Records

A program can open a record for reading. It can access it directly, as if it was just another chunk of memory. To write to a record, the application opens the record for writing. To do the actual writing, it calls a Data Manager routine to copy from another memory chunk to the record.

In this application, when a record is read, its data are copied into a `CTicTacGame` object. Data are written copying from a `CTicTacGame` object. When a game is opened, the board is displayed with the position as it was when the game was last closed.

CTicTacDatabase

This class handles the database access for the application. The declaration appears in **Listing 1**. It handles only one database.

Listing 1: Declaration of CTicTacDatabase

CTicTacDatabase

```
class CTicTacDatabase
{
protected:

    class CTicTacGame *mGame;
    static DmOpenRef sOpenRef;

public:

    static Boolean Open();
    static void Close();
    static UInt16 Count();
    static void GetGame ( Int16 inRecordNumber,
        CTicTacGame *outGame);
    static void SetGame ( Int16 inRecordNumber,
        CTicTacGame *inGame);
    static Int16 Add ( CTicTacGame *inGame );
    static void Delete ( Int16 inRecordNumber );

    CTicTacDatabase ();
    virtual ~CTicTacDatabase ();
};
```

Listing 2 shows the functions to save and retrieve the current game.

Listing 2: Definition of ::GetGame and ::SetGame

CTicTacDatabase

```
void CTicTacDatabase :: GetGame ( Int16 inRecordNumber,
    CTicTacGame *outGame )
{
    // Open the database
    if ( Open() )
    {
        // Get the numbered record and lock it
        MemHandle dataHandle = DmGetRecord ( sOpenRef,
            inRecordNumber );
        MemPtr dataPointer = MemHandleLock ( dataHandle );

        // Copy the data
        MemMove ( (void*)outGame, dataPointer, sizeof ( CTicTacGame ) );

        // Unlock release the record
        MemHandleUnlock ( dataHandle );
        DmReleaseRecord ( sOpenRef, inRecordNumber, false );
        // Close the database
        Close();
    }
    else
        outGame->Clear();
}

void CTicTacDatabase :: SetGame ( Int16 inRecordNumber,
    CTicTacGame *inGame )
{
    // Open the database
    if ( Open() )
    {
        // Get the numbered record and lock it
        MemHandle dataHandle = DmGetRecord ( sOpenRef, inRecordNumber );
        MemPtr dataPointer = MemHandleLock ( dataHandle );

        // Copy the data
        DmWrite ( dataPointer, 0, inGame, sizeof ( CTicTacGame ) );

        // Unlock release the record
        MemHandleUnlock ( dataHandle );
        DmReleaseRecord ( sOpenRef, inRecordNumber, true );
        Close();
    }
}
```

PREFERENCES

The word *preferences* is a little misleading. This means that the data that is used to store information that the application

Your Data Isn't an Important Part of the Job — It *IS* the Job.

VXA® FireWire

The High-Performance Tape Storage Solution For Apple Users

*"I can stick
my entire
hard drive
onto a tape
7 times
—that's just
plain cool!"*

Other backup media, such as CD-RWs and DVD-RAMs, simply can't compare to VXA-I tape technology when it comes to capacity, transfer rate and overall price. Fast, economical and easy-to-use, VXA FireWire offers:

Ultimate File Security

- 100% Data Restore and Interchange
- Plug-and-Play Connectivity

Sharable Media

- Multi-Gigabyte File Transport
- Portable, Hot-Pluggable

Real Time Digital Video

Cross-Platform Compatibility

- FireWire/IEEE 1394/iLink Supported By All Major Manufacturers



Technology	Capacity in MBs	Transfer Rate in MB/sec	Price per MB
VXA Tape	33000	3	\$0.03
InationTravan	10000	1	\$0.05
DVD RAM	9400	2.7	\$0.05
CD RW	700	1.9	\$0.50
Zip	250	1.2	\$0.76

LOWEST
COST per MB
OF ANY
Technology!

Call Exabyte sales at 1-800-774-7172
or visit us on the web at www.exabyte.com



Tape Storage By
VXA® **Exabyte®**

Conferences January 6 – 10, 2003

Expo January 7 – 10, 2003

San Francisco The Moscone Center

www.macworldexpo.com

January 6 – 10

Macworld
Conference & Expo.



Macworld Conference & Expo is recognized as the "must-attend" event for the Mac community. Join tens-of-thousands of attendees this January to benefit from high-level education and enjoy the strong community and camaraderie that takes place at this all-in-one marketplace.

"The best single source for information on the Mac. Even Windows users would be enlightened."

— Ken, President

"If you use a Mac, you are not using it to its fullest potential until after you attend Macworld."

— Michael, Graphic Artist



Flagship Sponsors

Macworld

Macworld.com



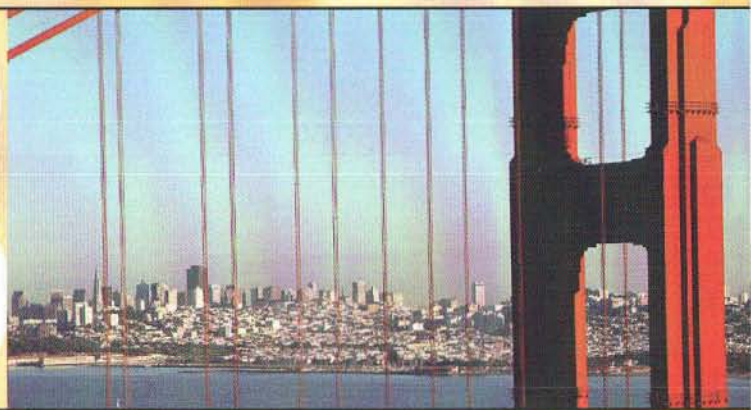
MacCentral

Platinum Sponsor

COREL

SF03

www.macworldexpo.com



Acquire what you need (knowledge, products, services or solutions) to stay competitive and on the cusp of technology.

Enjoy the one-stop-shop atmosphere only Macworld Conference & Expo can provide you.

Test drive brand new products and services — be one of the first to kick the tires of the latest innovations.

Apply knowledge learned from 5 intense educational days at Macworld Conference & Expo immediately.

Network, exchange ideas and build contacts with like-minded, or not so like-minded, users and industry gurus.

Feel what it is like to be part of a loyal, powerful and holistic community.

Discuss your issues, mention your concerns, or praise the manufacturers of your favorite products directly.

Register online with Priority Code: A-MTJ

For more information, call toll free 1-800-645-EXPO

needs to restore its state. Each time the user switches to a new application, the newly opened application needs to start where it left off the last time the user switched out of it.

For example, suppose the user switches out of the application while the Game Info form is displayed. The user may have been in the process of entering a new name. This partially entered new game name needs to reappear when the application is opened again. The case is similar for a selection made in the scrolling list in the Game List form. Let's see how this occurs.

CTicTacPreferences

The state of the current game is preserved in the application database when the application is switched out. This includes the state of the game. **Listing 3** shows the declaration for the application task to deal with preferences.

Listing 3: Declaration of CTicTacPreferences

```
class CTicTacPreferences
{
protected:
    static CTicTacPreferences *sPreferences;
    struct PreferencesRecord
    {
        Int16 mCurrentRecord;
        Int16 mSelectedRecord;
        Int16 mLastFormID;
        GameNameType mUnconfirmedName;
    };
    PreferencesRecord mPreferencesRecord;

public:
    CTicTacPreferences();
    ~CTicTacPreferences();
    static Int16 GetCurrentRecord();
};
```

CTicTacPreferences

```
static void SetCurrentRecord ( Int16 inRecord );
static Int16 GetSelectedRecord();
static void SetSelectedRecord ( Int16 inRecord );
static Int16 GetLastForm();
static void SetLastForm ( Int16 inFormID );
static void GetUnconfirmedName ( GameNameType outGame );
static void SetUnconfirmedName ( GameNameType inGame );
};
```

Sequence of Events

When the user activates another application, the system sends an `appStopEvent` to the current application. The main event loop picks up the event and exits. Control goes back to `TicTacPalmMain`, which calls `AppStop`. `AppStop` closes the active forms. As each form is closed, a `frmCloseEvent` is sent to it.

`AppStop` is defined in **Listing 4**. The function first closes all forms and deletes the `CTicTacPreferences` object. Then it deletes the objects that handle user action. As each form is deleted, a `frmCloseEvent` is generated. The handler for the form saves the current state of the form in the preferences data. Then, when `AppStop` deletes the preferences, the preference data are written to disk.

Listing 4: Definition of AppStop

```
static void AppStop(void)
{
    // Make sure the fields in each form are saved.
    FrmCloseAllForms ();

    if ( fPreferences )
    {
        delete fPreferences;
    }

    // Destroy the wrapper objects for forms.
}
```

AppStop

Who has the best fixed wireless solution in the industry?



There is no comparison!

Effective Data Throughput ⁽¹⁾ <small>(not to be confused with Signaling Rate)</small>	9.5 Mbps			Dynamic Bandwidth Control ⁽²⁾	Yes
Range ⁽²⁾	Fade margin	SU	SU-EXT	Latency Control	Yes
	6 dB	7 mi.	29 mi.		
	12 dB	3.3 mi.	10 mi.		
Users/AP	500			RF Thresholding ⁽⁴⁾	Yes
Non-Overlapping Channels	6			User Interface	Telnet, HTTP, Serial
Software Controlled Antenna Polarization	Yes			Price - SU/AP ⁽⁵⁾	\$495/\$895

(1) Measured on SmartBits 600 platform with 1518 byte sized packets.

(2) Trango specifies ranges with a 12 dB fade margin for added robustness.

(3) Indicates the ability to dynamically change up/down stream data throughput for unmatched bandwidth efficiency.

(4) A feature to minimize RF interference not found in competing systems.

(5) Low volume; ask us about higher volume discounts.

Trango Broadband is committed to providing the best technology at the most affordable price for your broadband wireless access deployment. Check us out and see why our Access5800™ solution is the best in class. Give us a call...Don't settle for less anymore!

trangobroadband
WIRELESS
A Division of Trango Systems, Inc.

Call Now!

858-653-3900

EMAIL: sales@trangobroadband.com
www.trangobroadband.com

FIXED WIRELESS...THAT WORKS!



(Ask about our third party leasing program)


```

if ( fGameBoardForm )
{
    delete fGameBoardForm;
    fGameBoardForm = NULL;
}

if ( fGameInfoForm )
{
    delete fGameInfoForm;
    fGameInfoForm = NULL;
}

if ( fGameListForm )
{
    delete fGameListForm;
    fGameListForm = NULL;
}
}

```

CGameInfoForm::Close

When the system executes `FrmCloseAllForms`, the system sends a close event to each form. This event will be processed by the `Close` function for the Game Info form. That function, shown in **Listing 5**, saves the partially entered game name in the preferences.

Listing 5: Definition of ::Close

```

CGameInfoForm::Close
Boolean CGameInfoForm :: Close()
{
    GameNameType name;
    GetFieldText ( GameInfoNameFieldField, name );
    CTicTacPreferences :: SetUnconfirmedName ( name );
    // Return false to tell the OS to clean up the form
    // in the usual way after we have extracted the info.
    return false;
}

```

CTicTacPreferences Destructor

When the data in the `CTicTacPreferences` object are up-to-date, `AppStop` calls the destructor for the preferences object, which then stores its data, as shown in **Listing 6**.

Listing 5: Destructor for CTicTacPreferences

```

CTicTacPreferences::~CTicTacPreferences()
{
    Boolean saved = true; //To be backed up at HotSync
    void *data = (void*)&mPreferencesRecord;
    UInt16 dataSize = sizeof ( PreferencesRecord );
    PrefSetAppPreferences (appFileCreator, appPrefID,
        appPrefVersionNum, data, dataSize, saved );
    sPreferences = NULL;
}

```

CONCLUSION

Storing application data is relatively easy on the Palm OS, as long as the data takes less than 64k. Restoring the state of the application using Preferences data requires some thought. Both would be easier if there were an application framework to handle the messy details.

REFERENCES AND CREDITS

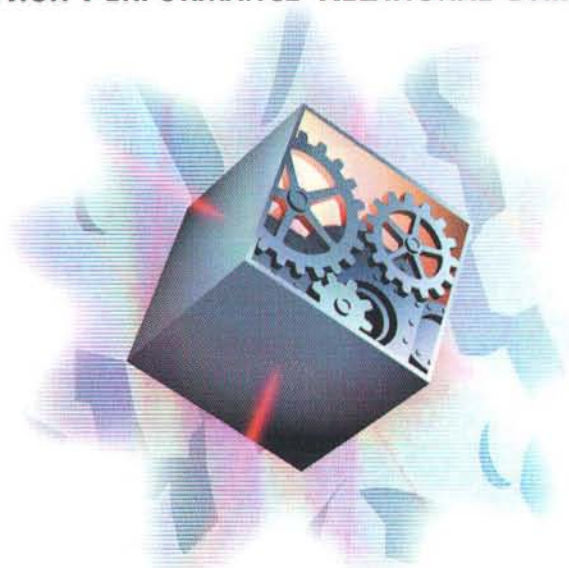
The Palm web site contains tons of information and links to related sites: <http://www.palmos.com/dev/>.

Thanks to Victoria Leonard for graphic resources. Thanks to Bob Ackerman, Mark Terry and Victoria Leonard for reviewing the text.

THE #1 RELATIONAL DATABASE ON MAC OS X

OPENBASE SQL

HIGH-PERFORMANCE RELATIONAL DATABASE



THE DATABASE THAT PAYS YOU CASH

*Announcing unlimited single-user
runtimes for just \$350 a year.*

Distribute your software with a royalty-free single-user version of OpenBase SQL. Receive monthly revenue when your users upgrade to one of our low-cost multi-user subscription licenses.

Go to www.openbase.com/cashback for more details!

Visit our Web site and check out the innovative features that make OpenBase SQL the database of choice for developers.

www.openbase.com



CREATE WITH POWER™

By Dan Wood, Alameda CA

Building the Internet-Connected Application

Integrating the Internet into your Ideas

INTRODUCTION

You may have noticed that it's almost impossible to run an application on Mac OS X without it trying to connect to the Internet to do something. The operating system checks for updates to itself. iTunes queries a database when you insert a music CD. OmniGroup's applications have a menu that says "Check for updates." The list goes on.

Taking advantage of the fact that many Mac OS X installations have a permanent (or frequent) Internet connection is something that you can have your own application do as well, even if it's not primarily categorized as an "Internet Application." The purpose of this article is to give you some motivation to do so, and some hints about making it a good experience.

ADVANTAGES OF A WELL-CONNECTED APPLICATION

So what are some of the benefits of an application that connects the Internet? Here are just a few:

- You can make sure that your users are aware of new versions of your program being available. This is a great boon for support, because you will have to deal with users of older versions of your application less frequently if they are spoon-fed the newest version.
- Individual components of your application—plug-ins, templates, etc.—can be listed and downloaded directly from the application if you maintain a repository of these items. Watson, for example, allows users to download and install additional plug-ins directly from the program, rather than having to go hunt for them all over the Internet.
- Data about the user (what version of the application and OS she is running, for example) can be collected by a server.

DUMB SERVERS

OK, so you think it's a good idea to have a server going for your desktop application can connect to, but what if you aren't a "server-side" kind of person? Are you going to have to learn new technologies like WebObjects or PHP? Will you need to buy an XServe and host at an expensive colocation facility?

That depends. You can actually accomplish quite a bit without any server-side programming, and very basic Web deployment capabilities. How's that? Just keep it simple, and put all of your logic on the client—the desktop program—and only data on the server.

Imagine the following scenario: You want to have your program connect to your server and determine if a new version of the application is available, so you can inform the user that they need to fetch the newest version. One approach is to send a message to your server and include the version of the application in that message. The server would then compare that value with the known latest version, and then either send back a message saying that the application is up to date, or a message saying that the user needs to upgrade. This, of course, would require programming on the server.

What if, instead, the client application just connected to the server and asked for the latest application version number. The client would then compare its own version with the number it received, and then decide whether or not to present an alert about application update availability.

This is the fundamental difference between a "smart" and a "dumb" server. The dumb server can merely be a static file that is served by your Web-hosting site of choice. If you're a Cocoa or Carbon programmer, you don't need to write code in an unfamiliar environment and find an Internet hosting service to host that server-side program for you.

Obviously, the Dumb Server approach will only go so far; some transactions are going to require logic on the server to react to parameters sent by the user. But here are some items

Dan Wood once took an introductory Arabic class, but nobody in the room knew what language they were being taught. He likes to buy fruits and vegetables from the farmer's market on Tuesday mornings. He missed the last two days of WWDC this year due to the birth of his son. He is the author of Watson, an application written in Cocoa. Dan thanks Chuck Pisula at Apple for his technical help with this series, and acknowledges online code fragments from John C. Randolph, Stéphane Sudre, Ondra Cada, Vince DeMarco, Harry Emmanuel, and others. You can reach him at dwood@karelia.com.

Free Mac® OS X!



... now, for a limited time, when you buy MYOB AccountEdge 3, we'll give you Mac OS X absolutely FREE.

Evolving with the Mac since 1989
Small Business Management and Accounting

www.myob.com/us



Small Business. Smart Solutions.™

800-322-MYOB (6962)

Offer details available online at www.myob.com/us. Valid for retail product #meurm purchased between 1/7/03 and 3/31/03. Cannot be combined with other offers. Void where prohibited. MYOB, the MYOB logo and AccountEdge are registered trademarks of MYOB Ltd. Mac and Mac OS X are trademarks of Apple Computer registered in the U.S. and other countries. © MYOB 2003

that can be put into a static file on the server, that your client program can react to.

- Current application version number, as discussed above. The client compares it to its own version and presents a message, accordingly.
- URL of where to fetch the new version of the application. If an update is needed, the client could download the update directly.
- Description of new features. To motivate the user into downloading the upgrade, you could provide a small list of new features.
- Data that may be needed later in the execution of the application. If your program allows the user to view a list of online downloadable modules, templates, or documents, you could provide a list of all available files along with the other information, and tuck it away until it is actually needed. Such a list could also contain version numbers, so your application could make sure it has the most up-to-date versions of the available components.
- News and status. There's an interesting way to stay connected with your user base, by providing a bit of news that might display each time the application is launched. It could be a "tip of the day" that you maintain on the server; it could be a holiday greeting, it could be links to relevant Web sites.

You could even provide different variations of the information you provide, so users running your program in "demo" mode will view one type of information, while your paid users get different data. Or your data could depend on the OS version used so Mac OS X 9 users will see one message while Mac OS X users get a different one. You could either put all variants into a single file, and let the application pick out the appropriate piece; or you could have the application access different URLs depending on that status. For example, you might serve up two files on your site, and your application would fetch the appropriate one

- http://www.mykillerapp.com/hello_unregistered.xml
- http://www.mykillerapp.com/hello_registered.xml

If your application is to be localized into different languages, you could also localize your messages as well. The client would request a file based on the user's primary preferred language (e.g. "hello_fr" for a French version). If that file doesn't exist, the client would then try again with the secondary language. If you don't think you'll be constantly adding new languages, you might want to put a limit on the number of attempts for your client to try, and store the best available language as a preference for the next time through.

SMART SERVERS

So you think you need a little bit of logic on the server?

Welcome to the world of client-server programming. You will need to determine just how much logic is going to go into your server and what kind of software and hardware you will need to run it. Unless you are building an application with high bandwidth and database needs, you can probably get by with a simple setup that is hosted on a service that costs about \$10 per month and provides server-side capabilities such as PHP and MySQL or Postgres. Macintosh has some useful links at the following URL: <http://www.macintosh.com/dotmacalt.html>. You could get more sophisticated with WebObjects or other technologies, deployed on colocated hardware, but that is way beyond the scope of this article.

A smart server has a number of advantages; it can provide whatever information is appropriate to the parameters given in a request; if it is told your operating system, language, registration status, etc., it can deliver exactly the message you wish to deliver. It can deliver personalized, targeted information, as well. For example, your client application could send a request for a "Web postcard" to be sent, and provide the recipient's name and email address; your server could build up an image and email the postcard. This wouldn't be possible without some logic on the server.

USER INTERACTION

If you are building Internet connectivity into your application, you need to make sure that the program behaves nicely. If the user doesn't have network connectivity, they might just be on a PowerBook, or they may have chosen to disable net access, perhaps if they are on a dialup connection. Just fail gracefully; you will probably have the opportunity to connect to the server later. Your program shouldn't nag the user too much if a new version is available; allow the user to stop telling you about the updates, or only show the reminder every so often. And don't initiate big downloads without the user's permission. They may be on a slow connection, and prefer to download things later.

GETTING USER DATA

It may be useful for you to pay attention to usage patterns of your application. Either for practical purposes, or if you are just curious, it can be useful to know something about your users, such as what version of the application they are using, or what version of the OS they are on. A "smart" server could take parameters passed from the server and add those data to a database. Even if you're just using a dumb server, you could perform some analysis on your web logs.

Be careful about privacy concerns! It is possible to have your client read user information (name, email) from the system settings, get addresses and phone numbers out of the address book, upload files in the user's home directory, and all sorts of unspeakable acts. The users of your program need

to trust that you aren't writing "spy-ware." You should be liberal with the privacy statements in your program, and if your program does upload any information about individual users, you should be very clear about how that information is used and stored.

ANTICIPATING FUTURE NEEDS

In creating your interaction model between your client and your server, you need to think about the directions that your application might be going, even if you don't plan on implementing any of these features for a while. Keep in mind that there may be thousands of users hitting your server for years to come, and they may be running version 1.0 of your software.

For example, you might want to include support for paid application upgrades. If you might want to charge a certain amount for version 2.0 or 3.0 of your program, you might want to provide upgrade price data, even if it won't be needed for a while.

Consider the possibility that your server may temporarily be unavailable to your users. What are the ramifications of that? You might want to provide a "backup server" URL in your application to try if the primary server doesn't work. Even if your primary server were "smart," you could have a backup server be just a static web page indicating system status, so your user would automatically get the latest news about your server if they were to connect and your primary server was unavailable.

Another consideration is your servers changing URLs. Imagine that you expand so much that you need to move your server, currently on your home IP address, to a different domain or subdomain. If you provided for a "forwarding address" in your server response, you could easily move people to your new domain. You should consider setting up a different subdomain to handle your server-side requests (even if it's just the same as your web server domain) so you could split it into a different machine in the future.

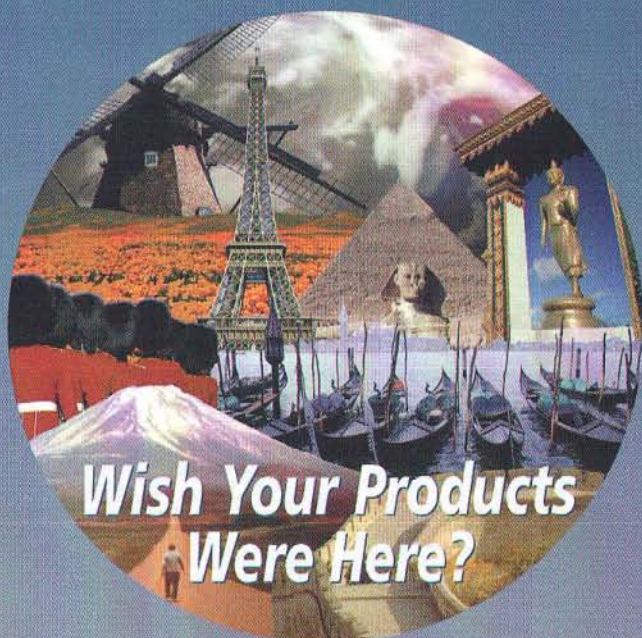
Finally, be sure to think about version compatibility between any plug-in or document files you may serve to your users. If you provide plug-ins, but your API changes between versions of your applications, you want to make sure that your users don't download plug-ins that won't work with your current application. If you might have a paid upgrade in the future, you want to make sure that users of your earlier versions of your program can still access their version 1.0-compatible plug-ins.

INFORMATION PROTOCOLS

Now that you have decided to include some Internet connectivity into your application, how should you go about transferring information back and forth?

Before going much further, you should consider firewall issues. Most users have no problem accessing Web sites through Port 80, so unless you have some compelling need to do

Classic Or Cocoa Applications? We Localize Them All!



- Translation • Engineering
- Localization • Project Management
- Desktop Publishing • Quality Assurance

To receive your FREE copy of
**The Guide to Translation and
Localization - Preparing Products
for the Global Marketplace:**

Call: **1-800-878-8523**

Fax: **1-503-419-4873**

Email: **info@lingosys.com**

Or visit: **www.lingosys.com**



15115 SW Sequoia Pkwy. #200 Portland, OR 97224

SOFTWARE LOCALIZATION MADE *easy*

POWERGLOT FEATURE HIGHLIGHTS

- LOCALIZE CLASSIC, CARBON™, COCOA® AND PALM OS™ APPS
- LEVERAGE EXISTING TRANSLATIONS
- AUTOMATE WITH APPLESCRIPT®
- IMPORT /EXPORT TRANSLATION MEMORIES

www.powerglot.com
browse, translate, click, done.

PowerGlot Software - Localization tools for Mac® OS
www.powerglot.com
info@powerglot.com

Mac OS, Carbon, Cocoa and AppleScript are trademarks of Apple Computer, Inc.
Palm OS is a trademark of Palm, Inc.

otherwise, you might as well go through that port. Naturally, if you will be transmitting confidential information, you're better off using SSL.

And unless there is a need for any real-time or open-ended connectivity, HTTP is a great protocol for simple request/response transfer. Your client makes a request, possibly sending some information along as parameters, and your server responds with what is effectively a Web page. Your client could perform a POST or GET; a GET is simpler because you can build up a simple URL of the form `http://www.host.com/filepath?param1=value1`. (Be sure to "escape" the special characters so that your URL is formed legally!) In Cocoa, you would just use `NSURLHandle` to fetch the contents of that URL. (If you need to use POST, Cocoa developers will need to dip down into Core Foundation or make use of `CURLHandle` (MacTech, Feb. 2002), available at `http://curlhandle.sourceforge.net/`.)

The response that your server returns should be formatted as simply as possible. You could return HTML or plain text, but that would require some parsing on your server's end. You could return any XML format of your choosing, and your client would then parse the XML stream.

One trick that makes it especially easy for Cocoa applications to get at the data returned from the server is to format the file on the server as an XML property list! Even if your server is running Linux or Windows and knows nothing of this format, there's no reason why you can't place a "plist" file on your web server and have that streamed to the client. You could maintain and edit this file on your Mac, to make sure it's a readable format, and then upload it to your server when it changes. (See Listing 1 for an hypothetical example file. Note that we are able to place arbitrary HTML within our XML by surrounding it with the `<![CDATA[...]]` directives.)

Listing 1: popcorn_hello.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST
1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>

<key>CurrentAppVersionNumber</key>
<string>1.0.7</string>

<key>CurrentAppFeatures</key>
<string>This update adds sound effects, and fixes a crash
when searching in Japanese.</string>

<!-- Below is only shown to unregistered users -->

<key>Unregistered News</key>
<string><![CDATA[

<font face="Lucida Grande">Thanks for trying out Popcorn!
<p>
Be sure to try out the new <b>Garlic Butter</b> module from
the <b>Popcorn &gt; Install More Plug-ins...</b> menu.
</font>
```



```
]]></string>
```

<!-- Below is only shown to REGISTERED users -->

```
<key>Registered News</key>
<string><![CDATA[
```

```
<font face="Lucida Grande">As a thank-you for our
registered users, we are offering free downloads of
<a href="http://www.karelia.com/peanutbutter/">PeanutButter
1.0</a> for a limited time. Give it a try!
</font>
```

```
]]></string>
```

```
</dict>
</plist>
```

The client application can convert the data stream it gets from the server into a Cocoa object (generally an NSDictionary or NSArray) by adding this category method to NSData, shown in Listing 2. (Constructing the @interface section is an exercise left to the reader.)

Listing 2: NSData+plist.m

```
propertyListFromXML;

Create a property list given XML data retrieved from a server.

@implementation NSData( plist )

// Return a property list from the data. Functions similarly to
// [NSDictionary dictionaryWithContentsOfFile:path]

- (id)propertyListFromXML
{
    CFPropertyListRef plist;
    CFStringRef errorString = nil;

    plist = CFPropertyListCreateFromXMLData(
        NULL,
        (CFDataRef)self,
        kCFPropertyListImmutable,
        &errorString);
    if(errorString)
    {
        NSLog(@"Error loading from Property List: %@",
            (NSString*)errorString);
        CFRelease(errorString);
    }
    return [(id)plist autorelease];
}
```

CONCLUSION

That about wraps up our little discussion on Internet connectivity. There are many other avenues that could be explored by innovative developers, but with just a little bit of work, it's possible to keep your users up-to-date.



MARX[®] CRYPTO-BOX USB Security Key For Macintosh

Use the CRYPTO-BOX[®] USB to secure data, networks, or software from unauthorized use.



Encryption

Access Control

License Control

User Limits



Supports all Macintosh Systems with USB including OS 8.6-9.1, & MAC OS X (CFM, Mach-O, and apps in classic), Windows in MAC OS 9 and Virtual PC 4. Metrowerks CodeWarrior C/C++ and Apple Project Builder C/C++ samples available

Contact

www.marx.com

MARX Software Security

2900 Chamblee-Tucker Road N.E., Bldg. 9
Atlanta, GA 30341, USA

☎ 1-800-MARX-INT

☎ 1-770-986-8887

fax 1-770-986-8891

info@marx.com

MARX Software Security GmbH

Vohburger Strasse 68
D-85104 Wackerstein
Germany

☎ (+49) 8403/9295-0

fax (+49) 8403/1500

contact-de@marx.com

MARX[®]
Software Security

Securing the Digital World[™]

By Brent Simmons

Writing Contextual Menu Plugins for OS X, part 2

Running Unix commands and handling text selections

In part one of this article (August 2002) we built a simple OS X contextual menu plugin that works in the Finder and provides a **Copy Path** command. It demonstrated the basics of implementing the COM-based interfaces for contextual menu plugins.

In this article we won't revisit the COM interface or other basics of contextual menu plugins—instead we'll go further, show how to run a Unix command from a contextual menu, how to send an update event to the Finder when a file changes, how to handle text selections, how to create a submenu with multiple commands, how to open a URL in a browser, and more.

The project file and source, built using the OS X 10.2 Developer Tools, can be downloaded from <http://ranchero.com/downloads/mactech/SamplePlugin.sit>.

PLUGIN OVERVIEW

This plugin (imaginatively named **SamplePlugin**) will provide three commands:

1. **Touch**—when one or more files (or folders) are selected, a Touch command will appear in the contextual menu. It will run the Unix touch command, which sets the modification date of the selected files to the current date and time.
2. **Copy**—when text is selected, this command will appear in a **Text Samples** submenu. It copies the selected text to the clipboard. (Note that while some applications already supply a Copy command in their contextual menu, many do not, and it's useful to have this command.)
3. **Search with Google**—when text is selected, this command will appear in a **Text Samples** submenu. It runs a search for the selected text in Google in one's default Web browser.

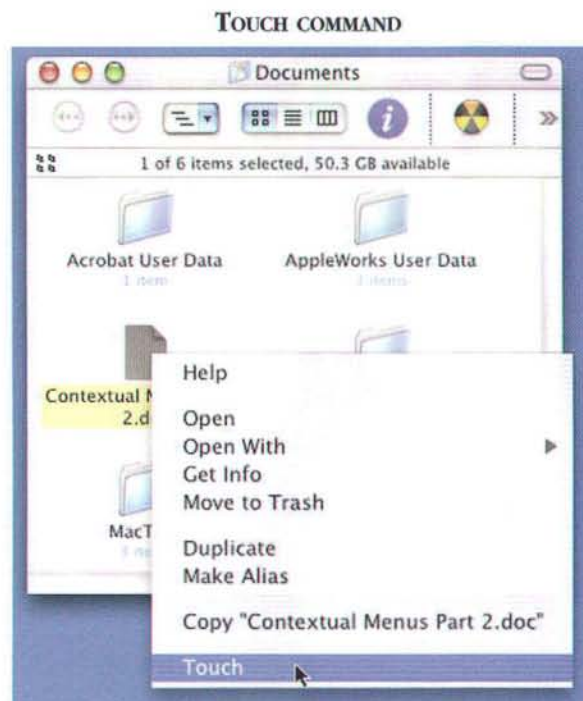


Figure 1. Touch command in Finder's contextual menu

The `examineContext` function in the plugin is called to give the plugin a chance to add commands to the contextual menu that's about to be displayed. This plugin does one of three things:

1. It adds a Touch command if one or more files or folders are selected.
2. It adds a Text Samples submenu, with Copy and Search with Google commands, if text is selected.
3. It adds no commands if neither files nor text is selected.

Listing 1: checking the context

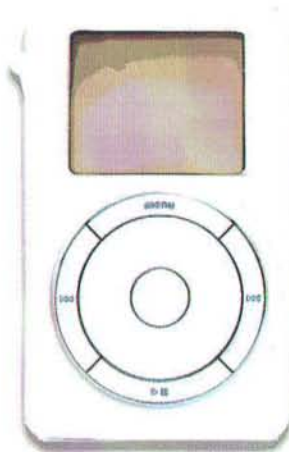
examineContext

```
static OSStatus examineContext (void *pluginInstance,
```

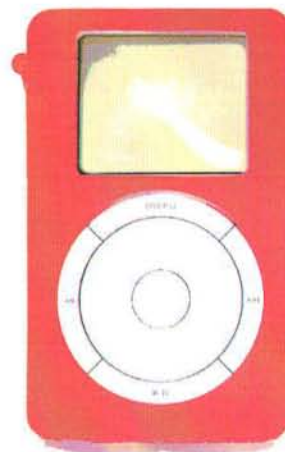
Brent Simmons is a Seattle-based independent Mac OS X developer and writer. He runs a Mac developer news weblog at ranchero.com; he can be contacted at brent@ranchero.com.



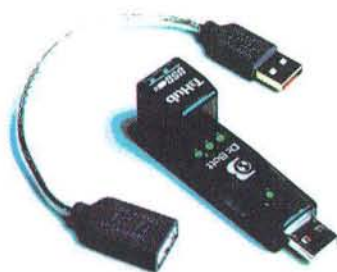
Asphalt iSkin
Protect your iPod in style



White Frost iSkin
Protect your iPod in style

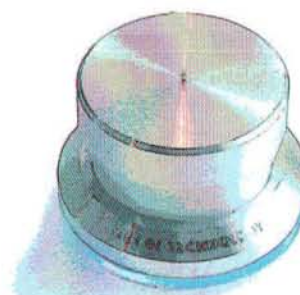


Hot House iSkin
Protect your iPod in style



T3 USB Hub
Tiny 3-port hub

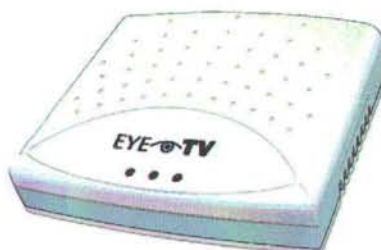

Dr. Bott
www.drbott.com
 Contact your local reseller
 to purchase Dr. Bott products.
 877.611.2688 503.582.9944



PowerMate
USB controller



ProScope
USB microscope



EyeTV
Watch and record



Half Keyboard
One-handed typing


```

const AEDesc *context, AEDescList *commandList)
{
    if (isFileOrListOfFiles (context))
        addFileCommandsToMenu (commandList);
    else if (isTextSelection (context))
        addTextCommandsToMenu (commandList);
    return (noErr);
}

```

`examineContext` first checks to see if one or more files or folders is selected. If so, it adds the file commands (the Touch command) to the contextual menu.

If the selection is not files or folders, then it checks to see if text is selected. If so, then it adds the Text Samples submenu and the Copy and Search with Google commands.

If neither of the above are true, it just returns `noErr`, having added no commands to the contextual menu.

The check to see if files are selected first checks to see if a single file or folder is selected. If not, then it checks to see if multiple files or folders are selected.

Listing 2: checking if files are selected

isFileOrListOfFiles

```

static Boolean isFileOrListOfFiles (const AEDesc *desc)
{
    if (isFileOrFolder (desc))
        return (true);
    return (isListOfFiles (desc));
}

```

`isFileOrFolder` checks a single `AEDesc` to see if it refers to a file or folder.

Listing 3: checking an `AEDesc` to see if it's a file or folder

isFileOrFolder

```

static Boolean isFileOrFolder (const AEDesc *desc)
{
    return (descIsOfType (desc, typeFSRef));
}

```

If the descriptor type is of `typeFSRef`, or can be coerced to `typeFSRef`, it's a file or folder, and so it returns true. `descIsOfType` performs this check. It's general—it can check for types other than `typeFSRef`. (In fact, this plugin also calls `descIsOfType` to check for a text selection.)

Listing 4: checking the type of an `AEDesc`

descIsOfType

```

static Boolean descIsOfType (const AEDesc *desc,
    OSType desiredType)
{
    AEDesc tempdesc = (typeNull, NULL);

    if ((*desc).descriptorType == desiredType)
        return (true);

    if (AECoeerceDesc (desc, desiredType, &tempdesc)
        == noErr) {
        AEDisposeDesc (&tempdesc);
        return (true);
    } /*if*/

    return (false);
}

```

Why check for `typeFSRef` instead of `typeAlias` or `typeFSS`? Because Apple's message is that `FSRefs` are the way to go in OS X. It would probably work as well to substitute one of these other types—but when your platform vendor tells you to do it a certain way, it's probably a good idea to listen.

Lists of files

If it's not just one file or folder selected, `isFileOrListOfFiles` next checks to see if it's a list of files or folders selected.

`isListOfFiles` loops through each item in the `AEDescList` and calls `isFileOrFolder` for each item. If any calls to `isFileOrFolder` return false, then the selection therefore contains items other than files or folders, and so `isListOfFiles` returns false.

Listing 5: looping through a list

isListOfFiles

```

static Boolean isListOfFiles (const AEDesc* desc)
{
    long numitems, i;
    OSErr err;

    err = AECountItems (desc, &numitems);
    if (err != noErr)
        return (false);

    for (i = 1; i <= numitems; i++) {
        AEKeyword keyword;
        AEDesc tempdesc = (typeNull, NULL);
        Boolean flFile = true;

        err = AEGGetNthDesc (desc, i, typeWildcard,
            &keyword, &tempdesc);
        if (err != noErr)
            return (false);

        flFile = isFileOrFolder (&tempdesc);
        AEDisposeDesc (&tempdesc);
        if (!flFile)
            return (false);
    }

    return (true);
}

```

`AECountItems` gets the number of items in the list. Then a for loop visits each item, calling `isFileOrFolder` with each. If `isFileOrFolder` returns false, then `isListOfFiles` returns false.

Otherwise it returns true, and then `isFileOrListOfFiles` returns true, and we're back to `examineContext`, which then calls `addFileCommandsToMenu` to add the Touch command to the contextual menu.

Adding the Touch menu command

To add the Touch command, this function calls `pushCommand` with the title of the command, its command ID, and the command list that will become the contextual menu that appears. The final parameter sent to `pushCommand` is `NULL`—if this item had a submenu it would be specified in the last parameter.

Listing 6: adding the localizable Touch command

addFileCommandsToMenu

```

static Boolean addFileCommandsToMenu

```



```
(AEDescList *commandList)
{
    CFStringRef touchCommand =
        CFCopyLocalizedString (CFSTR("Touch"),
            "Touch Menu Text");

    return (pushCommand (touchCommand, touchCommandID,
        commandList, NULL));
}
```

Note that it gets the name of the Touch command by calling `CFCopyLocalizedString`. This looks up the name in `MenuNames.strings` (a resource that's included with the project you downloaded). This way you can easily localize the plugin by editing a strings file—which is definitely preferred to hard-coding the names of menu items in the source code.

We'll otherwise skip `pushCommand` for now—we'll talk about it later when we show how to build submenus. For now just know that it adds the Touch command to the contextual menu.

`touchCommandID` is defined in `SamplePlugin.h`. The command ID, since it's just used internally, does not have to be localized, of course.

After this function returns, `examineContext` returns `noErr`, and the system handles displaying and tracking the contextual menu.

Handling a command

`handleSelection` is called by the system to actually run a command chosen by the user.

Listing 7: handling the user's command

`handleSelection`

```
static OSStatus handleSelection (void *pluginInstance,
    AEDesc *context, SInt32 commandID)
{
    switch (commandID) {

        case touchCommandID:
            runTouchCommand (context);
            break;

        case copyCommandID:
            runCopyCommand (context);
            break;

        case searchCommandID:
            runSearchCommand (context);
            break;

    }
    return (noErr);
}
```

Based on the command ID associated with the given command, it calls the corresponding function. If Touch is chosen, it calls `runTouchCommand`. There are similar cases for the Copy and Search with Google commands (more about those later).

Running the Touch command

The goal of this function is to pass to the system a string of text as if typed on the command line. We're calling the Unix `touch` command with one or more files or folders as arguments. The string that gets passed to the system will look something like this:

From PowerBook to PerfectBook



Order Now from PowerBook Depot
www.DevDepot.com



RADTECH™
Science that makes sense.
www.radtech.us

The RadTech family of solutions for protecting, ruggedizing and enhancing your Apple hardware

ScreensavRz

Display Protection / Cleaning /
Refinishing System
For all Notebooks!

Wildeepz

Display Cushion Upgrade
Improve fit and function. Eliminate
screen damage.

TiGlide/iGlide

G4 PowerBook / iBook Chassis
Enhancement Kit
Smooths tight hinges, *FAST*

PowerSleevez

Elegant Sleeve Cases
Fits your 'Book, like a glove.


```
"/usr/bin/touch "/path/to/some/file" "/path/to/some/other/file"
```

Each separate file path is enclosed in quotes because there may be spaces in the name.

So the first thing `runTouchCommand` does is get the list of selected files as a string suitable for passing to the system as command line arguments. Then it actually calls the system to run the Touch command. Finally it sends an update event to the Finder so it updates its display.

Listing 8: running the command

runTouchCommand

```
static void runTouchCommand (const AEDesc *desc)
{
    CFStringRef commandLineParams;

    if (!getFileListAsText (desc, &commandLineParams))
        return;
    callSystem (CFSTR("/usr/bin/touch"), commandLineParams);
    CFRelease (commandLineParams);
    sendUpdateEventToFinder (desc);
}
```

`getFileListAsText` is somewhat similar to the `isListOfFiles` function—it loops through the selected files in the same way. It also double-checks that each item actually is a file or folder.

Listing 9: getting the list of selected files as text

getFileListAsText

```
static Boolean getFileListAsText (const AEDesc *desc,
    CFStringRef *fileList)
{
    long numitems, i;
    OSStatus err;
    Boolean flSuccess = false;
    CFMutableStringRef s = CFStringCreateMutable
        (kCFAllocatorDefault, 0);

    if (s == NULL)
        return (false);

    err = AECCountItems (desc, &numitems);
    require_noerr (err, getFileListAsText_fail);

    for (i = 1; i <= numitems; i++) {
        AKeyword keyword;
        AEDesc tempdesc = {typeNull, NULL};
        Boolean flFile = false;

        err = AECGetNthDesc (desc, i, typeWildcard, &keyword,
            &tempdesc);
        require_noerr (err, getFileListAsText_fail);

        flFile = (tempdesc.descriptorType == typeFSRef);
        if (!flFile) {
            err = AECoerceDesc (&tempdesc, typeFSRef,
                &tempdesc);
            require_noerr (err, getFileListAsText_fail);
            flFile = (tempdesc.descriptorType == typeFSRef);
        }

        if (!pushFileAsText (&tempdesc, s))
            flFile = false;
        AEDisposeDesc (&tempdesc);
        if (!flFile)
            return (false);
    }

    flSuccess = true;

getFileListAsText_fail:
    if (flSuccess)
        *fileList = CFStringCreateCopy
```

```
(kCFAllocatorDefault, s);
else
    *fileList = NULL;

CFRelease (s);
return (flSuccess);
}
```

One of the parameters, `fileList`, is a pointer to a `CFStringRef` that will contain the list of files as quote-enclosed arguments. For each item, `pushFileAsText` is called, which adds each item to a `CFMutableString` created at the top of this function.

At the end of the function the `CFMutableString` is copied to the `fileList` parameter, which is a non-mutable `CFString`. (That is, if everything went well. In case of a problem the function returns false and `fileList` is NULL.)

Adding a single file to the list

`pushFileAsText` takes one `AEDesc` that references a file or folder, gets its Unix path, escapes double quotes in the path, escapes \$ characters in the path, puts double quotes around the path, adds a space, then adds the text to the passed-in `CFMutableString`.

Listing 10: adding a single file

pushFileAsText

```
static Boolean pushFileAsText (const AEDesc *desc,
    CFMutableStringRef s)
{
    CFStringRef pathString, escapedPathString;

    if (!getPathStringFromFSRef (desc, &pathString))
        return (false);

    if (!escapeShellText (pathString, &escapedPathString)) {
        CFRelease (pathString);
        return (false);
    }

    CFStringAppend (s, CFSTR("\""));
    CFStringAppend (s, escapedPathString);
    CFStringAppend (s, CFSTR("\" "));

    CFRelease (pathString);
    CFRelease (escapedPathString);
    return (true);
}
```

It calls `getPathStringFromFSRef` to get the path to the file as a `CFString`.

Then it escapes quotes and \$ characters inside the path string—if there are quotes anywhere in the path they must be escaped, otherwise there would be quote mis-matches in the command line text. \$ characters must be escaped to avoid variable interpolation. (For instance, if you had a file named \$USER for some strange reason, the system would replace \$USER with your actual user name. We don't want that to happen here.)

It then adds a quote to the mutable string, then adds the escaped path, then adds another quote and a space. You end up with a string like `"/path/to/some/file"` or `"/path to some/file"` or `"/path to/\$some\" file/."`

Getting the path string

`getPathStringFromFSRef` gets a Unix path string from an `FSRef` and puts it into a `CFString`.

Listing 11: getting a path string from an `FSRef`

`getPathStringFromFSRef`

```
static Boolean getPathStringFromFSRef (const AEDesc *desc,
                                       CFStringRef *pathString)
{
    FSRef fileRef;
    Size dataSize = AEGGetDescDataSize (desc);
    OSErr err;
    CFURLRef fileURL;

    err = AEGGetDescData (desc, &fileRef, dataSize);
    if (err != noErr)
        return (false);

    fileURL = CFURLCreateFromFSRef (kCFAllocatorDefault,
                                    &fileRef);
    if (fileURL == NULL)
        return (false);

    *pathString = CFURLCopyFileSystemPath (fileURL,
                                           kCFURLPOSIXPathStyle);
    return (*pathString != NULL);
}
```

First it gets the `FSRef` object from the `AEDesc` by calling `AEGGetDescData`. `CFURLCreateFromFSRef` gets a `CFURLRef` from the `FSRef`—and from that it gets the Unix-style path by calling `CFURLCopyFileSystemPath`.

`getCFString` is a simple wrapper around `CFStringCreateWithCString`, which creates a `CFString` based on a C string.

Escaping text for the command line

Back to `pushFileAsText`: it next calls `escapeShellText`. It does a couple find-and-replace operations in a `CFString`. It escapes double quotes and `$` characters to avoid quote mis-matches (when a file contains a double quote in its name) and variable interpolation (on the off chance you have a file named something like `$USER`).

Listing 12: `escapeShellText`

`escapeShellText`

```
static Boolean escapeShellText (CFStringRef source,
                                CFStringRef *dest)
{
    CFStringRef tempString;
    Boolean flSuccess = false;

    if (!replaceAll (CFSTR("\""), CFSTR("\\\""),
                    source, &tempString))
        return (false);
    flSuccess = replaceAll (CFSTR("$"), CFSTR("\\$"),
                           tempString, dest);
    if (flSuccess)
        CFRelease (tempString);
    return (flSuccess);
}
```

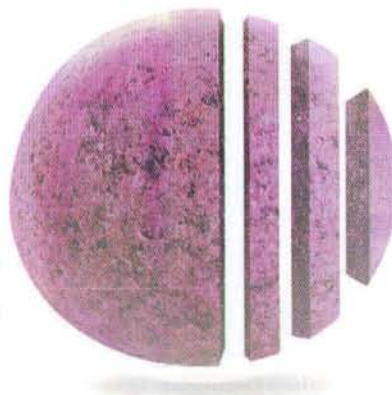
`replaceAll` actually performs the replacements.

FREE UPGRADES FOR LIFE!

TIRED OF BEING SHAKEN DOWN FOR EXPENSIVE, BUGGY, FORCED UPGRADES FROM THE BIG VENDORS? JOIN THE PEOPLE WHO BELIEVE THAT YOU SHOULD BUY SOMETHING ONCE – AND HAVE IT FOREVER! SUPPORT AN OPEN AND FREE SOFTWARE DEVELOPMENT COMMUNITY NOT RUN BY MARKETING TYPES OR BEAN COUNTERS.



The applications in Stone Studio — starring Create® — have all the most used features of these applications: Illustrator, Quark Xpress, DreamWeaver, Freehand, Corel Draw, PageMaker, Timeslips, Acrobat Writer, Stuffit Deluxe, ImageReady, iPhoto and more. Visit www.stone.com/Stone_Studio_Successes to see how people just like you get their best ideas out into the world.



STONE STUDIO™

7 APPS 1 LOW PRICE 0 PAID UPGRADES

STONE STUDIO DOWNLOAD NOW FROM WWW.STONE.COM



\$299

Listing 13: replaceAll

replaceAll

```
static Boolean replaceAll (CFStringRef searchFor, CFStringRef
replaceAll, CFStringRef source, CFStringRef *dest)
{
    CFMutableStringRef mutableString;
    CFRange range;

    mutableString = CFStringCreateMutableCopy
        (kCFAllocatorDefault, 0, source);
    if (mutableString == NULL)
        return (false);

    range = CFRangeMake (0,
        CFStringGetLength (mutableString));
    CFStringFindAndReplace (mutableString, searchFor,
        replaceWith, range, 0);

    *dest = CFStringCreateCopy (kCFAllocatorDefault,
        mutableString);
    CFRelease (mutableString);
    return (*dest != NULL);
}
```

The first parameter to `replaceAll` is the string to find, the second parameter is the replacement string, the third parameter is the source string in which to search, and the last parameter is a pointer to a `CFStringRef` that will contain the result.

Aside: CFStrings

Question: why use `CFStringRef`s and `CFMutableStringRef`s? Why not traditional C strings or Pascal strings?

Because, as with `FSRefs`, Apple's message is that `CFStrings` are the way to go. But beyond that they have several benefits:

1. They're easy to manipulate. Functions like `CFStringAppend` make it very easy, for instance, to add text to a string. Check out `CFString.h`—there is a gold mine of functions which make string manipulation easy.

2. `CFStrings` are "toll-free bridged" with Cocoa `NSStrings`. This means that, among other things, when writing Cocoa code, you can call `CFString` functions with `NSStrings` as parameters. This is an example of a kind of convergence, where you can write code that works in Carbon apps as well as Cocoa apps.

3. `CFStrings` take some of the headache out of supporting various character encodings. Support for Unicode and other text encodings is an important part of a modern operating system, and `CFStrings` can store Unicode text as well as other encodings.

Back to runTouchCommand

Now it's time to actually call the system to run the Touch command.

If `getFileListAsText` succeeds, `runTouchCommand` calls `callSystem`, which takes two parameters: the path to the command to execute and the command line parameters (as a single string).

The path to the Touch command is `/usr/bin/touch`. The command line parameters in this case is the list of files created in `getFileListAsText`.

Note the line (in `runTouchCommand`): `callSystem (CFSTR("/usr/bin/touch"), commandLineParams);`

The `CFSTR("/usr/bin/touch")` part is a shortcut for creating a `CFStringRef` from quoted text. (Cocoa developers will note that it's the equivalent of typing `@"/usr/bin/touch"` to create an `NSString`.)

Calling the system

It's simple to call the system to execute a command-line string. There's a function actually named `system` that takes a C string and executes the command as if typed in the Terminal.

(The man page for `system` is pretty short and worth checking out.)

Listing 14: calling the system

callSystem

```
static void callSystem (CFStringRef command,
    CFStringRef params)
{
    char *buffer;
    CFMutableStringRef fullCommand;

    fullCommand = CFStringCreateMutableCopy
        (kCFAllocatorDefault, 0, command);
    if (fullCommand == NULL)
        return;

    CFStringAppend (fullCommand, CFSTR(" "));
    CFStringAppend (fullCommand, params);

    if (!getCString (&buffer, fullCommand,
        kCFStringEncodingUTF8))
        goto callSystem_exit;

    printf ("System call: ");
    printf (buffer);
    printf ("\n");

    system (buffer);

    callSystem_exit:
    CFRelease (fullCommand);
    if (buffer != NULL)
        free (buffer);
}
```

Our `callSystem` function takes two `CFStringRef` parameters—the path to the command and the command-line parameters. In order to call the system command it needs to create a C string of this form: `/path/to/command param1 param2 param3`.

In this plugin it will look something like this: `/usr/bin/touch "/path/to/file1" "/path/to/anotherFile"`

The `fullCommand` string is a mutable `CFString`. It starts by copying the command (the `/usr/bin/touch` part) to itself.

Then it appends a space, since there needs to be a space between the command and the parameters. Then it appends the parameters.

At this point we have the string in the right form for the system, except that it's a `CFString` rather than a C string. So it calls `getCString` to get a C string.

Listing 15: getCString

getCString

```
static Boolean getCString (char **cStringToGet,
    CFStringRef cfString, CFStringEncoding encoding)
{
    UInt32 lenText = sizeof (UniChar) *
        CFStringGetLength (cfString) + 1;

    *cStringToGet = malloc (lenText);
    if (*cStringToGet == NULL)
        return (false);

    if (!CFStringGetCString (cfString, *cStringToGet,
        lenText, encoding)) {
        free (*cStringToGet);
        *cStringToGet = NULL;
        return (false);
    }

    return (true);
}
```

This function is a wrapper for `CFStringGetCString`, which gets a C string from a `CFString`. First this function allocates a buffer that's the length of the `CFString` plus one for zero-termination. Then it calls `CFStringGetCString`. If that call fails, the buffer is freed and the function returns false. If all's well it returns true.

Back to callSystem

Before calling the `system` function, notice the trio of `printf` commands. An important point: one way to debug contextual menus is to use `printf` commands. The output appears in the Console. (Which lives at `/Applications/Utilities/Console`.) If you launch it, then run the Touch command from a contextual menu, you'll see a copy of what gets passed to the `system` command in the Console.

Though not used here, another important function, similar to `printf`, is the `CFShow` command. This prints a description of a CoreFoundation object—such as a `CFString`—to the Console. (See `CFShow` and `CFShowStr` in `CFString.h` and in the documentation.)

Okay, now it finally calls the system with the C string from `getCString`. This runs the Unix touch command with the list of files as parameters.

This function cleans up, then control goes back to `runTouchCommand`, then back to `handleSelection` which returns `noErr`.

That's it for running a Unix command. You know the basics—you can do just about anything Unix-y from a contextual menu plugin now.

But it didn't work...

Oh yes it did. But it might not have appeared to work.

Here's why: sometimes the Finder doesn't update its display right away when you make a change to a file from a contextual menu. And so it may look like the command didn't work, even though it actually did.

So `runTouchCommand` sends the Finder a `KAESync` event via the `sendUpdateEventToFinder` function:

Use as a first priority, not as a last resort...



Data Rescue recovers your valuable
data before, during or after
a hard disk crisis.

PROSOFT
engineering inc.

www.prosofteng.com

1.866.428.3282



© 2002 Prosoft Engineering Inc. All rights reserved.
Data Rescue is a trademark of Prosoft Engineering Inc.
Mac and the Mac logo are trademarks of Apple Computer, Inc., in the U.S.
and other countries. The "Built for Mac OS X" graphic is a trademark
of Apple Computer, Inc., used under license.
All other trademarks are the property of their respective owners.

Listing 16: sending a kAESync event to the Finder

sendUpdateEventToFinder

```
static void sendUpdateEventToFinder (const AEDesc *desc)
{
    AEResourceDesc finderAddressDesc = {typeNull, NULL};
    AppleEvent event = {typeNull, NULL};
    AppleEvent reply = {typeNull, NULL};
    OSType finderSignature = 'MACS';
    OSStatus err;

    err = AEResourceDesc (&finderAddressDesc, &finderSignature,
        sizeof (finderSignature), &finderAddressDesc);
    if (err != noErr)
        return;

    err = AEResourceDesc (&event, kAEFinderSuite, kAESync,
        &finderAddressDesc, kAutoGenerateReturnID,
        kAnyTransactionID, &event);
    require_noerr (err, sendUpdateEventToFinder_fail);

    err = AEResourceDesc (&event, keyDirectObject, desc);
    require_noerr (err, sendUpdateEventToFinder_fail2);

    AEResourceDesc (&reply, kAENoReply + kAECanInteract,
        kAENormalPriority, kAEDefaultTimeout, NULL, NULL);

    sendUpdateEventToFinder_fail2:
    AEResourceDesc (&event);

    sendUpdateEventToFinder_fail:
    AEResourceDesc (&finderAddressDesc);
}
```

It creates an Apple event that just asks the Finder to update its display. (See `FinderRegistry.h` for this and other commands you can send the Finder.)

This function passes to the Finder, as the direct object parameter, the same `AEDesc` context received in `handleSelection`. One assumes that the Finder knows what to do with this `AEDesc` since it came from the Finder in the first place.

`kAENoReply` is specified, since a reply wouldn't be useful, and we don't really care if the Finder failed to handle the event. We hope it handled the event, but if it didn't it's no big deal, since the Touch command succeeded anyway.

TEXT COMMANDS AND SUBMENUS

Let's back all the way up to `examineContext` and show how to add a submenu to the contextual menu and how to handle our two text commands (Copy and Search with Google).

Recall that if it's not one or more files or folders that are selected, then `examineContext` checks to see if it's text that's selected. `isTextSelection` performs this check. If the check returns true, then `examineContext` calls `addTextCommandsToMenu` to add the submenu and its commands.

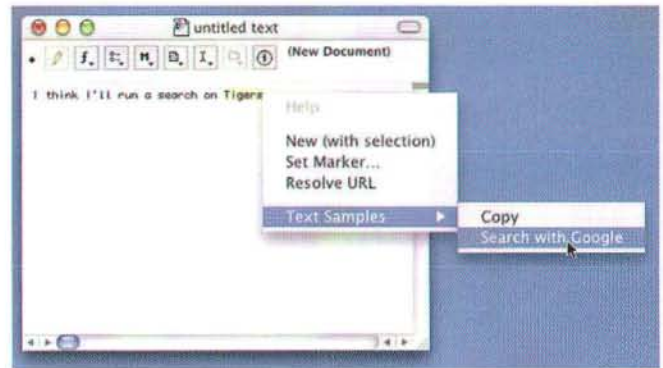


Figure 2. Text commands in BBEdit's contextual menu.

isTextSelection

This function checks the `AEDesc` that `examineContext` got from the system to see if it's of type text or can be coerced to type text.

Listing 17: checking if a selection is a text selection

isTextSelection

```
static Boolean isTextSelection (const AEDesc *desc)
{
    return (descIsOfType (desc, typeChar));
}
```

`typeChar`, defined in `AEDataModel.h`, is 'TEXT.' This function calls `descIsOfType`, which was also called by `isFileOrFolder` to determine if an `AEDesc` refers to a file or folder.

If `isTextSelection` returns true, then `examineContext` calls `addTextCommandsToMenu` with the `AEDescList` from the system that will become the contextual menu.

Adding text commands

The plugin will build a menu item named Text Samples with a submenu. The submenu will contain two commands: Copy and Search with Google.

Here things are done in what may seem like a backward order. First the submenu (Copy and Search with Google commands) is created, then the menu item is created that will contain the submenu.

Listing 18: adding text commands to the menu

addTextCommandsToMenu

```
static Boolean addTextCommandsToMenu (AEDescList*
    commandList)
{
    AEDescList submenuCommands = {typeNull, NULL};
    OSStatus err;
    Boolean flSuccess = false;
    CFStringRef textSubMenuName =
        CFStringCreateWithCString (CFSTR("Text Samples"),
            "Text Samples Menu Text");

    err = AEResourceDesc (NULL, 0, false, &submenuCommands);
    if (err != noErr)
        return (false);
}
```




REAL Software and MacTech present the REALbasic Showcase to highlight some of the fantastic solutions created by REALbasic users worldwide. The showcase illustrates the wide range of applications that developers using REALbasic can create. Some benefit any Mac user, and others are more specific. All of them are seriously cool!

REALbasic is the powerful, easy-to-use tool for creating your own software for Macintosh, Mac OS X, and Windows. It runs natively on Mac OS X as well as earlier versions of the Mac OS. For more information, please visit: [<www.realbasic.com>](http://www.realbasic.com).

The Made with REALbasic program is a cooperative effort between REALbasic users and REAL Software, Inc. to promote the products created using REALbasic and the people who create them. For more information about the Made with REALbasic program, please visit: [<www.realbasic.com/realbasic/mwrb/Partners/MwRbProgram.html>](http://www.realbasic.com/realbasic/mwrb/Partners/MwRbProgram.html).

Extend REALbasic with Advanced Plugins

Spreadsheet Controls:

We provide a wide range of spreadsheet controls for REALbasic, including multistyled and custom rendering spreadsheet controls.

A	B	C
This is some incredible list		
1	Some text	More text
2	Some text	More text
3	Some text	More text
4	Some text	More text
5	Some text	More text
6	Some text	More text
7	Some text	More text
8	Some text	More text

- More than 32k of rows.
- Classic, OS X and Win32.
- Accelerated for maximum speed.
- Images in cells.



Cryptography:

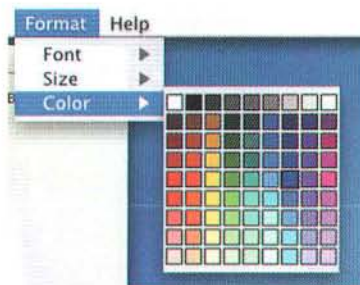
We provide data encryption, encoding, compression and hashing for REALbasic.

Supported Algorithms:

- Encryption:
 - e-CryptIt
 - BlowFish (448 bit)
 - AES (Rijndael) (256 bit)
- Encoding:
 - e-CryptIt Flexible
 - Base 64
 - BinHex
 - MacBinary III
 - AppleSingle / Double
 - UUcoding
- Compression:
 - Zip on Strings and streams (.gz)
- Hashing and Checksums:
 - CRC32 / Adler32
 - MD5 / HMAC_MD5
 - SHA / SHA1 / HMAC_SHA1

Other Plugins:

We have many other plugins for REALbasic, including plugins to do advanced MacOS Toolbox tasks and more custom Controls.



Speed up development and make more advanced applications by using plugins ! Get free demos at www.einhugur.com



Einhugur Software
sales@einhugur.com
www.einhugur.com



piPop

Pop-up Hierarchical
File Navigation and Launcher



TelnetLauncher

Bookmark and Launch your
Telnet and SSH sessions



SimpleKeys

Set your Function Keys
to type stuff for you!

piDog Software

<http://www.pidog.com/>



**MAC MUSIC MANAGEMENT FOR
SONY® 5 TO 400 DISC CD CHANGERS**



**Control up to 12 changers (4,800 music CDs)
Control Any Brand Stereo Receiver
Play MP3 and other Sound Files
Plus much, much more!!!**

www.titletrack.com

info@titletrack.com

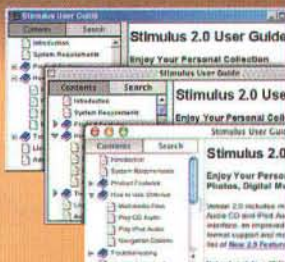
UniHelp™

Cross-Platform HELP Module for REALbasic

Apple Help Viewer, Microsoft HTML Help, WinHelp.... Why waste precious time developing a different Help system for each platform when you can easily use UniHelp for all of them? Compatible with Classic Mac, Mac OS X and Microsoft Windows 98-XP.

2002 Cubie Award Winner for Best Development Aid

- Displays HTML & Text Files.
- Supports URL Links, Relative Links, Anchors & Email Links.
- Parses More Than 20 HTML Tags & More Than 60 Special Characters.
- Supports Images, Video and Audio.
- Built-in Search Engine.
- Supports Context-Sensitive Help.
- Hierarchical Table of Contents.
- Familiar Help-style Interface.
- GUI Support for 6 Languages.
- Extremely Customizable.
- Small Memory Footprint.
- Easy to Use & Royalty-Free!



SPECIAL OFFER

Purchase UniHelp and Receive a FREE Upgrade to the 2003 Release of **HelpLogic 1.0**
The Help Authoring Solution

DOWNLOAD NOW
www.ebutterfly.com

electric butterfly

The Journal of Macintosh Technology & Development

MacTech

Got a great product built with REALbasic?

Promote your product through the very cost effective REALbasic Showcase in MacTech Magazine.

For more information, contact us at
adsales@mactech.com

iScreensaver Designer

the cross-platform solution for Macintosh and Windows



Version 3.0 featuring **OS X** coming soon!

- build both Macintosh and Windows screensavers with a single click, no matter what system* you are using!
- use any QuickTime 6.0 movie format : Macromedia Flash 5.0, MPEG, Cinepak, MP3, Midi, AVI, DV Video... or, now with version 3.0, build your own basic Slide Shows!
- include a hidden movie that can be unlocked with a registration code
- customize and fully-brand both Installers and Screensaver control panels with pictures and text
- Screensavers install without DLLs, extensions, or restarts

simple WYSIWYG editor

supports interactive Flash and QuickTime

consistent cross-platform user interface

try before you buy fully functional online downloads



The iScreensaver Designer editing environment

creating screensavers for both Windows and Macintosh has never been this easy

<http://iScreensaver.net>
email : info@iScreensaver.net

* supported systems, as of June 2002, include:
Macintosh OS 8.6 to 9.2.2, Microsoft Windows 95/98/ME, NT4/NT2000/XP
iScreensaver Designer version 3.0 will support up to Macintosh OS 10.2

©2000-2002 Xochi Media Inc. Made using REALbasic.


```

if (!buildTextSubmenu (&submenuCommands)) {
    AEDisposeDesc (&submenuCommands);
    return (false);
}

flSuccess = pushCommand (textSubmenuName, nil,
    commandList, &submenuCommands);
AEDisposeDesc (&submenuCommands);
return (flSuccess);
}

```

The submenu is, like the `commandList` passed to `examineContext` from the system, an `AEDescList` that will specify one or more commands. `submenuCommands` will hold the submenu: `buildTextSubmenu` creates this submenu.

Listing 19: building a submenu

buildTextSubmenu

```

static Boolean buildTextSubmenu (AEDescList *commands)
{
    CFStringRef searchCommand = CFCopyLocalizedString
        (CFSTR("Search with Google"), "Search Menu Text");
    CFStringRef copyCommand = CFCopyLocalizedString
        (CFSTR("Copy"), "Copy Menu Text");

    if (!pushCommand (copyCommand, copyCommandID,
        commands, nil))
        return (false);

    return (pushCommand (searchCommand, searchCommandID,
        commands, nil));
}

```

It calls `pushCommand` twice, with the command names, command IDs, and the `AEDescList` that will contain the commands.

If it succeeds, then `addTextCommandsToMenu` calls `pushCommand` to add the Text Samples menu item that will contain the submenu.

Here's the important point: both the main contextual menu and any submenus are `AEDescList`s. That means that you add commands the same way, whether it's the main menu or any submenus. `pushCommand` is thus a reusable piece of code, useful whether you're building one or more submenus or not.

Listing 20: adding a command to a menu

pushCommand

```

static Boolean pushCommand (CFStringRef commandName,
    SInt32 commandID, AEDescList* commands,
    AEDescList *submenuToAttach)
{
    OSStatus err = noErr;
    AERecord commandRecord = {typeNull, NULL};
    Boolean flSuccess = false;
    char *commandNameCString;

    err = AECreatelist (NULL, 0, true, &commandRecord);
    require_noerr (err, pushCommand_fail);

    if (!getCString (&commandNameCString, commandName,
        kCFStringEncodingUTF8))
        goto pushCommand_fail;

    err = AEPutKeyPtr (&commandRecord, keyAName,
        typeUTF8Text, commandNameCString,
        strlen (commandNameCString) + 1);
    free (commandNameCString);
    require_noerr (err, pushCommand_fail);

    if (commandID != NULL) {

```

```

        err = AEPutKeyPtr (&commandRecord,
            keyContextualMenuCommandID,
            typeLongInteger, &commandID, sizeof (commandID));
        require_noerr (err, pushCommand_fail);
    }

```

```

    if (submenuToAttach != NULL) {
        err = AEPutKeyDesc (&commandRecord,
            keyContextualMenuSubmenu, submenuToAttach);
        require_noerr (err, pushCommand_fail);
    }

```

```

    err = AEPutDesc (commands, 0, &commandRecord);
    if (err == noErr)
        flSuccess = true;

```

```

    pushCommand_fail:
    AEDisposeDesc (&commandRecord);
    return (flSuccess);
}

```

When using `pushCommand` to add a menu item that contains submenu, just call it with a `NULL` `commandID` and a non-`NULL` `AEDescList` that specifies the submenu it contains (`submenuCommands`, in this case).

When using `pushCommand` to add a menu item that does not contain a submenu, specify its command ID, and send a `NULL` `AEDescList`, since it will have no submenu.

When `addTextCommandsToMenu` is finished, control returns to `examineContext`, which returns `noErr`. Then the system displays and tracks the contextual menu with the Text Samples menu item and its submenu.

Handling the text commands

Back to `handleSelection`—if the command ID is the ID of the copy command, `runCopyCommand` is called. If it's the command ID of the search command, `runSearchCommand` is called. Let's do `runCopyCommand` first.

The Copy command copies the selected text to the clipboard (unsurprisingly).

Listing 21: running the Copy command

runCopyCommand

```

static void runCopyCommand (const AEDesc *desc)
{
    CFStringRef selectedText;

    if (!getTextFromDesc (desc, &selectedText,
        kCFStringEncodingMacRoman))
        return;
    writeStringToClipboard (selectedText);
    CFRelease (selectedText);
}

```

First it gets the selected text from the `AEDesc` via `getTextFromDesc`, then it writes that text to the clipboard via `writeStringToClipboard`, then it cleans up and returns to `handleSelection`.

getTextFromDesc

Any time you're handling text selections you need to get the selected text. `getTextFromDesc` gets it as a `CFStringRef`.



Listing 22: getting text from an AEDesc

getTextFromDesc

```
static Boolean getTextFromDesc (const AEDesc *desc,
    CFStringRef *text, CFStringEncoding encoding)
{
    long len = AEGetDescDataSize (desc);
    char *s;
    Boolean flSuccess = false;

    s = malloc (len + 1);
    if (s == NULL)
        return (false);

    AEGetDescData (desc, s, len);
    s [len] = '\0';
    flSuccess = getCFString (text,
        (const char *) s, encoding);
    free (s);
    return (flSuccess);
}
```

AEGetDescDataSize tells us how long the selected text is. It then allocates a buffer to get that text, then calls AEGetDescData to put it into the buffer. It zero-terminates the buffer so that it's a C string.

The function then calls getCFString to get a CFString based on the C string it got from the AEDesc.

After freeing the buffer (which is no longer needed) it returns true if the CFString was created and false otherwise.

getCFString is a simple wrapper around CFStringCreateWithCString.

Listing 23: getting a CFString from a C string

getCFString

```
static Boolean getCFString (CFStringRef *stringToGet, const
    char *cString, CFStringEncoding encoding)
{
    *stringToGet = CFStringCreateWithCString
        (kCFAllocatorDefault, cString, encoding);
    return (*stringToGet != NULL);
}
```

writeStringToClipboard

runCopyCommand then writes the text to the clipboard via writeStringToClipboard.

Listing 24: writing text to the clipboard

writeStringToClipboard

```
static void writeStringToClipboard (CFStringRef s)
{
    ScrapRef scrap;
    char *cString;

    if (!getCString (&cString, s, kCFStringEncodingMacRoman))
        return;
    ClearCurrentScrap ();
    GetCurrentScrap (&scrap);
    PutScrapFlavor (scrap, kScrapFlavorTypeText,
        kScrapFlavorMaskNone, strlen (cString),
        (const void *) cString);
    free (cString);
}
```

Note that this seems a little crazy—in getTextFromDesc we created a CFString from a C string, then in writeStringToClipboard we turn around and get a C string from a CFString. Why?

LOVE TESTING YOUR CODE?

DIDN'T THINK SO.

LET EGGPLANT TEST IT FOR YOU.

Test Automation for Mac OS X

Easy to learn and use

Powerful scripting

Faster testing

Test any application on any system
Mac, Windows, Unix, Linux

eggplant™

Test any application on any system.

Redstone

2695 Northpark Drive
Suite 201

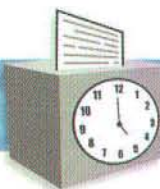
Lafayette, CO 80026

(800) 891-3486 or (720) 890-0211

info@redstonesoftware.com

www.redstonesoftware.com/eggplant

Time Track



Keep track of every second of your time and bill for it with Time Track! Built in instructions make it easy to use. It is a simple way to manage your billable time for multiple projects and create a web page to show to your clients. Only \$24.95 per single user license per platform. Finally! A versatile time tracking solution for Macintosh, Windows, and Palm!

www.trinfinitysoftware.com



By TLA Systems

The Dock with more than one dimension.

Create multiple docks of any size, assign hot keys and even put the Trash back on the Desktop. A flexible and feature-laden tool for power-users. Runs natively on Mac OS X and 9 in five languages.

"...you made the switch to OS X a lot easier for me..." - Bob LeVitus

"...DragThing can rightfully be called an indispensable aid to working with your Mac..." - MacUser UK

Download a copy now from www.dragthing.com.



Watson 1.6

Winner of the 2002 Apple Design Award for Most Innovative Mac OS X Product! An Aqua experience for the most useful Web services: TV & Movie Listings, Reference, Translation, Stocks, Flights, Package Tracking, and more. With an open architecture for third-party tools as well. Download the demo now!

www.karelia.com/watson



Eudora Internet Mail Server (EIMS) 3.1 is the latest version of the most popular Internet mail server for the Macintosh. If you need to handle email for a dozen users, or thousands of users, EIMS is a reliable and easy to use solution.

EIMS 3.1 is available for US\$400.00, there are no limits on the number of users that can be added, and free email support is included.

For more information, see
<http://www.eudora.co.nz/>

The Journal of Macintosh Technology & Development

MacTech

Got a great product sold through Kagi?

Promote your product through the very cost effective Kagi Showcase in MacTech Magazine.

For more information, contact us at

adsales@mactech.com

Well, in the interests of maintainability, I like having just one function that gets the selected text. Instead there could be one function that returns it as a CFString and another that returns it as a C string. I prefer getting a CFString because CFStrings are so easy to manipulate, and chances are you're going to want to do some operation on the selected text.

In this case, however, we're not manipulating the selected text in any way, just writing it to the clipboard.

Again, the `writeStringToClipboard` function could take a C string rather than a CFString, but my bet is that most of the time I'll have a CFString, since I use CFStrings so much.

In other words, what I've done is standardize on using CFStrings internally, so it's only at the edges—when getting text from an AEDesc, writing text to the clipboard, or calling the system function—where it's sometimes necessary to convert to or from a CFStringRef. (And, as you'll note later in the Search with Google command, there are cases where no conversion is necessary as more and more APIs take CFStrings.)

So `writeStringToClipboard` works in some ways like the `callSystem` command—it takes the CFStringRef and gets a C string. A difference here is what it does with it—it clears the current scrap (the clipboard), gets a reference to the current scrap, then writes the C string as text to the clipboard. Finally it frees the C string it got.

That's it for the Copy command. On to Search with Google.

Running the Search with Google command

This command gets the selected text then creates a search URL string based on a base URL. The search URL is then opened in one's default Web browser.

Listing 25: running the Search command

runSearchCommand

```
static void runSearchCommand (const AEDesc *desc)
{
    CFStringRef selectedText, urlString;

    if (!getTextFromDesc (desc, &selectedText,
        kCFStringEncodingMacRoman))
        return;

    if (!createEncodedSearchURLString
        (CFSTR(googleSearchURL), selectedText, &urlString)) {
        CFRelease (selectedText);
        return;
    }

    openInBrowser (urlString);
    CFRelease (selectedText);
    CFRelease (urlString);
}
```

As in `runCopyCommand`, it calls `getTextFromDesc` to get the selected text as a CFString.

Then it creates the URL string to pass to the browser by calling `createEncodedSearchURLString`. Then it calls `openInBrowser` with that URL string to open it in the browser. Then it cleans up and returns.

It's 3AM on Sunday

IS THE SERVER STILL RUNNING?

**End the
worry**

Kick-off!

**The simple reliability
solution for Mac OS 9 and X**



- System-level Crash Detection**
patented hardware-software integration
- Automatic Crash Recovery**
system restart or power cycle, as needed
- Monitor Custom Applications**
software developer's kit and plug-ins included
- Scheduled Restarts**
cures memory leaks and fragmentation
- Restart after Power Failure**
even if shut down by your UPS
- Comprehensive Error Logging**
to help you track down problems
- Simple Installation**
just plug in AC cord and USB cable



For all these features *plus* six power outlets controllable by phone tones, schedules and scripts, consider

www.sophisticated.com

SOPHISTICATED CIRCUITS INC.

Copyright ©2002 Sophisticated Circuits, Inc. Kick-off! and PowerKey are registered trademarks of Sophisticated Circuits, Inc. Mac OS is a registered trademark of Apple Computer, Inc.

Encoding a URL string

This function demonstrates some of the utility of CFStrings. It takes a string like `http://foo.com?q=` as the base URL and a string like "some selected text" as the search arguments. It creates a string that a Web browser would understand, as in `http://foo.com?q=some+selected+text`.

In our case the base URL is `http://www.google.com/search?q=` (defined in `SamplePlugin.h`), so the final URL string will look something like `http://www.google.com/search?q=some+selected+text`.

Listing 26: Encoding a URL string

```
createEncodedSearchURLString

static Boolean createEncodedSearchURLString (CFStringRef
baseURL, CFStringRef searchArgs, CFStringRef *dest)
{
    CFStringRef urlString, urlStringPlusEncoded;

    urlString = CFStringCreateWithFormat
        (kCFAllocatorDefault, NULL, ("%@"),
        baseURL, searchArgs);

    if (!replaceAll (CFSTR(" "), CFSTR("+"), urlString,
        &urlStringPlusEncoded)) {
        CFRelease (urlString);
        return (false);
    }

    *dest = CFURLCreateStringByAddingPercentEscapes
        (kCFAllocatorDefault, urlStringPlusEncoded, NULL,
        NULL,
        kCFStringEncodingISOLatin1);

    CFRelease (urlStringPlusEncoded);
    CFRelease (urlString);
    return (*dest != NULL);
}
```

First it concatenates the base URL and the search args via `CFStringCreateWithFormat`. (Cocoa developers, note how similar this is to `NSString's stringWithFormat` method.)

Then it calls `replaceAll` to replace spaces with + characters (which is what Web browsers and servers want us to do).

Then it's necessary to URL-encode the URL string—characters such as `é` and so on have to be converted to percent-encoding. `CFURLCreateStringByAddingPercentEscapes` does this.

Finally it cleans up, and `*dest` is the encoded string.

Opening a URL in the default Web browser

`runSearchCommand` then calls `openInBrowser` to actually open the URL in one's default Web browser.

Listing 27: opening a URL

```
openInBrowser

static void openInBrowser (CFStringRef urlString)
{
    CFURLRef urlRef = CFURLCreateWithString
        (kCFAllocatorDefault, urlString, NULL);

    if (urlRef != NULL) {
        LSOpenCFURLRef (urlRef, NULL);
        CFRelease (urlRef);
    }
}
```

The sole parameter is a `CFStringRef`, the encoded URL string created in `createEncodedSearchURLString`. The system call for opening a URL in a browser wants a `CFURLRef`, which is created via `CFURLCreateWithString`. (Cocoa developers: this is like `NSURL's URLWithString` method.)

Then the `LaunchServices` function `LSOpenCFURLRef` is called. This useful function also opens local URLs, doing different things depending on what kind of URL it has. In this case it's an HTTP URL, and so it opens the URL in the default browser. Note that the code doesn't even know or care what the default browser is: the system handles it for us. (Cocoa developers note how this is like `NSWorkspace's openURL` method.)

You can get more control over how the default browser opens the URL by calling `LSOpenFromURLSpec`—for instance, you can tell the browser not to come to the front. But in this sample we use `LSOpenCFURLRef` because the default behavior is what we want anyway.

After calling `openInBrowser`, `runSearchCommand` cleans up and returns control to `handleSelection`, which returns `noErr`.

ROOM FOR IMPROVEMENT

There some things this plugin could do better—but since this an article rather than a book, we'll leave that up to you. A couple obvious things:

When getting text to write to the clipboard or run a search on, it naively assumes MacRoman text encoding, which is not necessarily the case. It's unlikely the results will be satisfactory on, for instance, Japanese systems.

Another issue is error reporting—this plugin just fails silently when something goes wrong. At a minimum it could write an error message to the Console. But since most people don't leave their Console running, a better choice would be to display a dialog box letting the user know the error.

BONUS TIPS

Debugging

I mentioned earlier that `printf`, `CFShow`, and `CFShowStr` are useful for debugging—you can print to the Console to help you figure out what's going on.

But what if you want to really debug—that is, set breakpoints and step through your code? A plug-in isn't an application, so you can't just run and debug it as-is. Here's what you do:

From the `Project` menu choose `New Custom Executable`. Click the `Set...` button in the dialog that appears. Choose an application that supports system contextual menus. Me, I always use `Script Editor`, just because it launches very quickly on my machine. Click the `Finish` button.

Need some FIX IT help for Mac OS X 10.2?!

Mac FixIt founder Ted Landau offers disaster relief

Get Ted's Top 10
Disaster Relief
Free Tipsheet.
Simply visit
www.peachpit.com/mactech103/download

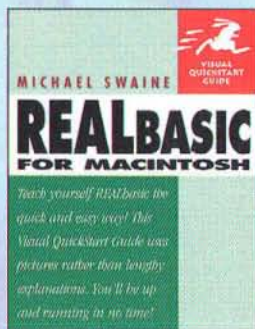


Mac OS X Disaster Relief,
Updated Edition
Ted Landau
0-321-16847-X • \$34.99

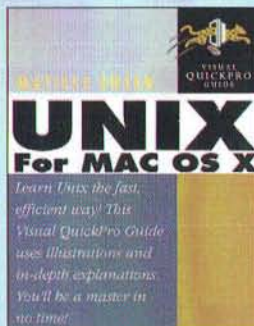
Head off disaster before it strikes with the latest edition of Mac problem-solving guru Ted Landau's troubleshooting manual for Mac OS X. Updated with a new appendix devoted to Mac OS X 10.2, the book covers Mac OS X 10.2's new features, the pros and cons of updating system software, Library directories and folders, files and font maintenance, crash prevention and recovery, printing and networking problems, Unix commands for OS X, and more.

Get With the Program!

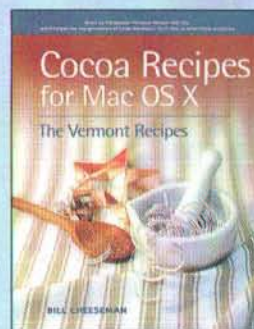
Three new books to help you get seriously hands-on with your Mac.



REALbasic for Macintosh:
Visual QuickStart Guide
Michael Swaine
0-201-78122-0 • \$21.99



Unix for Mac OS X: Visual
QuickPro Guide
Matisse Enzer
0-201-79535-3 • \$24.99



Cocoa Recipes for
Mac OS X
Bill Cheeseman
0-201-87801-1 • \$44.99

Save 15% plus free shipping on these titles and more!

Simply go to www.peachpit.com and enter coupon code EM-A3AA-MTMC at check out.



Peachpit Press

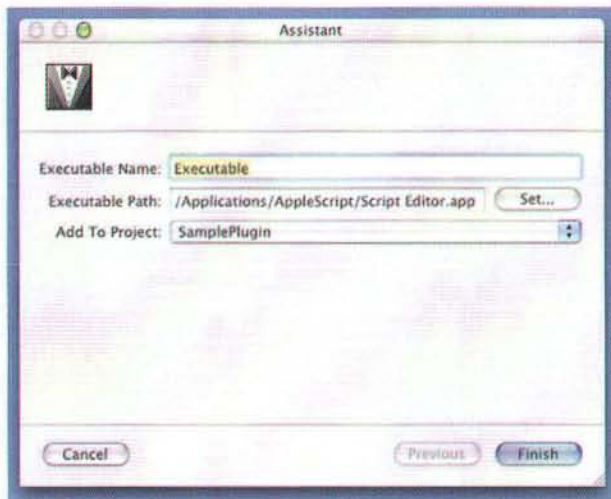


Figure 3. Attaching an executable to the plugin

Then, before debugging your plugin, set a breakpoint somewhere—in `examineContext`, for instance. To begin debugging, choose **Debug Executable** from the **Debug** menu. Script Editor (or whatever app you chose) will launch.

Type some text in the app you chose, select it, then ctrl-click (or right-click) on the selected text. Project Builder's debugger should come to the front with execution stopped at your breakpoint. From there you can debug as normal.

(Unfortunately I have not figured out how to get this to work with the Finder, but it works very well for other apps.)

So your cycle becomes like this: build your plugin, delete the old version from your `~/Library/Contextual Menu Items/` folder, copy the new build to your Contextual Menu Items folder, choose **Debug Executable**, then debug. Lather, rinse, repeat.

Quitting apps

Remember that, when you rebuild a plugin and put it in the **Contextual Menu Items** folder, you need to quit any app you're testing with in order to force it to reload the plugin. What's the fastest way to quit an app that's not in front? Ctrl-click (or right-click) on its icon in the Dock and choose **Quit** from the pop-up menu.

This doesn't work with the Finder. However, you can enable a **Quit** menu item for the Finder so that you can type `Cmd-Q` in the Finder. In Terminal, type: `defaults write com.apple.finder QuitMenuItem Enabled`

You will have to log out then log back in for the changes to take effect. If later on you want to disable the **Quit** menu item in the Finder, in Terminal type: `defaults write com.apple.finder QuitMenuItem Disabled`

Quit the Finder and re-start it (by clicking its icon in the Dock) and the **Quit** menu item will be gone.

CONCLUSION

To summarize a few important points from this article:

- To run a Unix command, call the `system` function with a C string, with text that looks like what you would type on the command line. Remember to add and escape quotes and \$ characters as needed.

- CFStrings are good to use for lots of reasons, but the best one is that it makes string manipulation easy.

- Whether adding a command to the main contextual menu or to a submenu, you're doing the same thing, adding an item to an `AEDesclList`. There's no difference.

- There are other high-level routines like `LSOpenCFURLRef` that do things like open a URL in the default browser. If you find yourself doing things like writing code to support different browsers, there's a good chance you're working way too hard. Pay special attention to the CoreFoundation and LaunchServices frameworks. Even Cocoa developers can benefit by looking at these frameworks, since there are useful functions there that have no Cocoa equivalent but that work with `NSStrings` and `NSURLs` and so on.

- `printf`, `CFSHOW`, and `CFSHOWSTR` are useful for debugging—but, even better, you can truly debug a plugin by attaching an executable and setting a breakpoint.

- Remember to quit apps you're testing with after rebuilding your plugin. You can even enable a **Quit** item for the Finder via the `defaults` Terminal command.

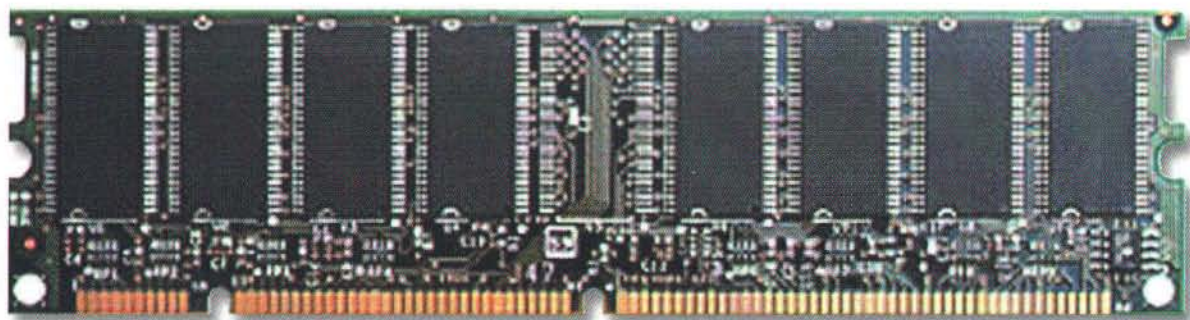
Thanks to...

For review and feedback for this article, thanks to Jim Correia, Quentin D. Carnicelli, and George Warner. Any remaining errors or oddities are mine.

REFERENCES

- Apple sample code: http://developer.apple.com/samplecode/Sample_Code/Human_Interface_Toolbox/SampleCMPlugIn.htm
- CoreFoundation Plugin Services: http://developer.apple.com/techpubs/macosx/CoreFoundation/PluginServices/Plug_in_ServicesConcTask/index.html
- CoreFoundation String Services: http://developer.apple.com/techpubs/macosx/CoreFoundation/StringServices/String_Services_ConcTask/index.html
- CoreFoundation URL Services: http://developer.apple.com/techpubs/macosx/CoreFoundation/URLServices/URL_Services/index.html
- Contextual Menu Workshop is a free framework for creating OS X contextual menus: <http://homepage.mac.com/tkukiel/cmworkshop.html>

There are some things in life you
can never have enough of...



Built-to-order. Lifetime Guarantee. Competitive Pricing.

**Come see us at Macworld
for GREAT Show Special Prices
Booth 3761 in the North Hall!**

**DEV
DEPOT®**

www.devdepot.com/memory

Voice: 877-DEPOT-NOW (877-337-6866) • Outside US/Canada: 805/494-9797
E-mail: orders@devdepot.com

List of Advertisers

AEC Software	39
Aladdin Knowledge Systems, Inc.	45
Aladdin Systems, Inc.	11
Asante Technologies, Inc.	13
Avesta Computer Services, Ltd.	19
Big Nerd Ranch, Inc.	44
DevDepot	28-29
Dr. Bott LLC	79
Ein hugur Programming Resources	70
Electric Butterfly	71
Eudora Internet Mail Server	75
Exabyte	37
FairCom Corporation	1
Felt Tip Software	23
Fetch Softworks	9
Full Spectrum Software, Inc.	31
IDG World Expo Corporation	50-51
James Sentman Software	71
Karelia Software	75
Lingo Systems	53
MacDirectory	7
Marx Software Security	52
Mathemaesthetics, Inc.	35
Microsoft	55
MYOB US, Inc.	15
/n software inc.	59
Netopia, Inc.	21
O'Reilly & Associates, Inc.	56
OpenBase International, Ltd.	17
Paradigma Software	27
Peachpit Press	77
Perforce Software, Inc.	81
piDog Software	70
PowerGlot Software	73
PrimeBase (SNAP Innovation)	63
Prosoft Engineering, Inc.	67
REAL Software, Inc.	69
REAL Software, Inc.	82
Redstone Software, Inc.	41
RiverSong InterActive	70
Runtime Revolution Limited	ifc2
Small Dog Electronics	33
Sophisticated Circuits, Inc.	43
Stone Design Corp.	65
Sybase, Inc.	2-3
The Software MacKiev Company	49
Thursby Software Systems, Inc.	61
TLA Systems Ltd.	75
Trango Broadband Wireless	8
Trinifinity Software	75
Utilities4Less.com	54
WIBU-SYSTEMS AG	25
Xochi Media Inc.	71
Zero G Software	47

List of Products

Accessories • DevDepot	28-29
Accessories • Dr. Bott LLC	79
Adobe Press • Peachpit Press	77
Big Nerd Ranch • Big Nerd Ranch, Inc.	44
Books • O'Reilly & Associates, Inc.	56
c-tree Plus • FairCom Corporation	1
Consulting & Training Services • Avesta Computer Services, Ltd.	19
Data Rescue • Prosoft Engineering, Inc.	67
DAVE • Thursby Software Systems, Inc.	61
Development & Testing • Full Spectrum Software, Inc.	31
DragThing • TLA Systems Ltd.	75
EIMS • Eudora Internet Mail Server	75
FastTrack • AEC Software	39
Fetch • Fetch Softworks	9
InstallAnywhere • Zero G Software	47
InstallerMaker, StuffIt • Aladdin Systems, Inc.	11
IntraCore Switches • Asante Technologies, Inc.	13
IP®Works! • /n software inc.	59
iScreensaver Designer • Xochi Media Inc.	71
Long Distance Phone Service • Utilities4Less.com	54
MacDirectory • MacDirectory	7
Macworld Conference & Expo • IDG World Expo Corporation	50-51
MYOB • MYOB US, Inc.	15
Office for OS X • Microsoft	55
OpenBase • OpenBase International, Ltd.	17
piDog Utilities • piDog Software	70
PowerGlot • PowerGlot Software	73
PowerKey Pro & KickOff! • Sophisticated Circuits, Inc.	43
PrimeBase • PrimeBase (SNAP Innovation)	63
QuickerHelp & Consulting • The Software MacKiev Company	49
REALbasic Plug-ins • Ein hugur Programming Resources	70
REALbasic • REAL Software, Inc.	82
REALbasic Showcase • REAL Software, Inc.	69
• Redstone Software, Inc.	41
Resorcerer • Mathemaesthetics, Inc.	35
Revolution • Runtime Revolution Limited	IFC2
SCM Software • Perforce Software, Inc.	81
Security • Marx Software Security	52
SmallDog.com • Small Dog Electronics	33
Software and Hardware • Aladdin Knowledge Systems, Inc.	45
Software Protection • WIBU-SYSTEMS AG	25
Sound Studio • Felt Tip Software	23
Stone Studio • Stone Design Corp.	65
SyBase • Sybase, Inc.	2-3
Timbuktu Pro & netOctopus • Netopia, Inc.	21
Time Track • Trinifinity Software	75
TitleTrack Jukebox • RiverSong InterActive	70
Translation & Localization • Lingo Systems	53
UniHelp Module • Electric Butterfly	71
Valentina • Paradigma Software	27
VXA • Exabyte	37
Watson • Karelia Software	75
Whistle Blower • James Sentman Software	71
Wireless Networking • Trango Broadband Wireless	8

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

To meet deadlines, developers have two choices:

1. Use Perforce
2. Cut Corners



For developers under pressure to manage source code and do more in less time, Perforce's Fast Software Configuration Management System is the must-have tool.

With rival SCM systems, the only way to quicken the pace is to cut corners - but in the long run you pay the price with missed deadlines, uncertain contents, buggy releases and no way back to previous builds.

With Perforce, the fast way is always the right way. Install it fast, learn it fast, execute operations fast. With other SCM systems, developers face an unpleasant choice: do it the right way or do it the fast way. Perforce's speed and reliability mean fast is right. See how Perforce compares with other leading SCM systems at <http://www.perforce.com/perforce/reviews.html>

Run at full speed even with hundreds of users and millions of files. At the core of Perforce lies a relational database with well-keyed tables, so simple operations can be accomplished in near-zero time. Larger operations (like labeling a release and branching) are translated into keyed data access, giving Perforce the scalability that big projects require.

Work anywhere. Perforce is efficient over high-latency networks such as WANs, the Internet and even low-speed dial-up connections. Requiring only TCP/IP, Perforce makes use of a well-tuned streaming message protocol for synchronizing client workspace with server repository contents.

Develop and maintain multiple codelines.

Perforce Inter-File Branching™ lets you merge new features and apply fixes between codelines. Smart metadata keeps track of your evolving projects even while they develop in parallel.

Truly cross platform. Perforce runs on more than 50 operating systems, including Windows and nearly every UNIX® variation, from Linux® and Mac OS® X to AS/400 and more.

Integrate with leading IDEs and defect trackers:

Visual Studio.NET®, Visual C++®, Visual Basic®, JBuilder®, CodeWarrior®, TeamTrack®, Bugzilla™, ControlCenter®, DevTrack® packages, and more.



Fast Software Configuration Management www.perforce.com

Download your free 2-user, non-expiring, full-featured copy now from www.perforce.com
Free (and friendly) technical support is on hand to answer any and all evaluation questions.

All trademarks used herein are either the trademarks or registered trademarks of their respective owners.

like ships
passing in
the night

ARE YOU HONEST, handsome, successful, financially secure, intelligent, world traveled, cultured, creative, fit, playful, adventurous, passionate, humorous, caring, loving, and between 46 and 58? Respond to European, Florida female court-

Come see us at Macworld in MacTech Central! Download a free demo. www.realbasic.com